卓越工程师教育系列“十二五”规划教材
高等学校公共课计算机规划教材

C 语言程序设计与实践

廖小飞　李敏杰　主编

许武军　白恩健　陈　雯　蒋学芹　副主编

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书是依据高等学校计算机类和信息类各专业基础课程教学的要求与目标而编写的理论与实践相结合的教材。本书以 C 语言基本知识和基本概念为引领，将知识融入各个实例，通过实践来学习 C 语言程序设计，注重 C 语言基本概念、基本编程思想的介绍和应用，始终贯彻“教、学、做”相结合的原则，使学生掌握 C 语言程序设计方法，能够学以致用，培养学生使用 C 语言来解决实际问题的能力。全书共 11 章，主要内容包括：计算机程序设计概述，C 语言基础，数据输入与输出，控制结构程序设计，数组和字符串，指针，函数，构造数据类型，编译预处理，文件，高质量编程规范。本书提供配套电子课件、程序代码和习题参考答案。

本书可作为高等学校计算机类和信息类各专业本科生或专科生“C 语言程序设计”课程的教材，或者其他专业、其他课程的参考书，也可以作为初学者学习 C 语言程序设计的入门教材，还可供有关工程技术人员学习、参考。

图书在版编目（CIP）数据

C 语言程序设计与实践 / 廖小飞，李敏杰主编. —北京：电子工业出版社，2015.8
卓越工程师教育系列“十二五”规划教材
ISBN 978-7-121-26220-3

I. ①C… II. ①廖… ②李… III. ①C 语言－程序设计－高等学校－教材 IV. ①TP312

中国版本图书馆 CIP 数据核字（2015）第 119345 号

策划编辑：王晓庆
责任编辑：王晓庆
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1092　1/16　印张：15.5　字数：447 千字
版　　次：2015 年 8 月第 1 版
印　　次：2015 年 8 月第 1 次印刷
印　　数：3000 册　　定价：35.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：(010) 88258888。

前　言

近几十年以来，计算机技术发展非常迅速，在各个行业都有着广泛的应用，已成为当今社会各行各业不可缺少的工具。软件技术是计算机技术的核心和灵魂，软件行业的发展水平和规模也成为衡量一个国家现代化程度和综合国力的重要标志。进行软件设计的程序设计语言非常多，现在全球大约有600多种编程语言，但流行的编程语言只有20几种，C语言自诞生之日起，由于具有功能丰富、使用灵活、运行速度快、能够操作硬件、应用范围广等优势，一直是最流行的程序设计语言之一，根据著名的TIOBE开发语言排行榜公布的结果，C语言几乎每月都处于第一名位置。

C语言程序设计是计算机类和信息类专业的专业基础课和必修课，也是这些专业学生入校后最先接触的一门专业课，其重要性和基础性不言而喻。通过该课程的学习，读者应掌握C语言的基本语法和程序设计的基本思想，并掌握传统的结构化程序设计的一般方法，培养严谨的程序设计思想、灵活的思维方式及较强的动手能力，并以此为基础，逐渐掌握复杂软件的设计和开发方法，为后续“数据结构”、“面向对象程序设计”等课程的学习打下扎实的理论和实践基础。

作为软件开发的入门课程，C语言程序设计有着非常重要的地位和作用。为了培养技术应用型人才，使学生掌握高级程序设计语言的知识，在实践中逐步掌握程序设计的思想和方法，提高学生使用C语言来解决实际问题的应用能力，我们为该课程编写了本书，适合高等学校低年级无程序设计基础的学生使用，帮助学生尽快掌握C语言，达到教学要求。该教材有如下特色。

（1）从零基础学习C语言程序设计，适合高等学校低年级无计算机基础或计算机基础较弱的学生学习。本书从最基本的编程思想开始，一步步指导学生如何编写程序、如何编译、如何运行和调试程序。对于C语言的学习也是一步步由浅入深地讲解，使得学生在较短时间内较快地掌握C语言程序设计的知识和方法。

（2）实例教学。本书的每个知识点都由实例构成，可以通过程序的运行结果来理解程序的原理，从而掌握每个知识点。通过实例学习相关知识，围绕模块进行教学和实训，降低了学生学习的困难度。

（3）涵盖C语言基础知识和开发实践，将课堂教学、实验上机、课程设计的内容进行一体化，通过理论教学和实践应用并重，从各个层面提高学生的程序设计能力，实现“教、学、做”合一。

本书以C语言程序设计为主线，从应用和实践出发，通过实例引入内容，重点讲解C语言程序设计的知识和方法。全书共11章，理论教学参考学时数为36～48学时，实验上机参考学时数为12～16学时，课程设计参考学时数为16～20学时，有关章节内容可根据专业要求和学时情况酌情调整。该教材可作为高等学校计算机类和信息类各专业本科生或专科生“C语言程序设计”的教材，或者其他专业、其他课程的参考书，也可以作为初学者学习C语言程序设计的入门教材，还可供有关工程技术人员学习、参考。教学中，可以根据教学对象和学时等具体情况对书中的内容进行删减和组合，也可以进行适当扩展。为适应教学模式、教学方法和手段的改革，本书提供配套电子课件、程序代码和习题参考答案等，请登录华信教育资源网（http://www.hxedu.com.cn）注册下载。

参加本书编写的有东华大学廖小飞和上海大学李敏杰，其中，廖小飞编写了第2、3、4、5、6、7、11章，李敏杰编写了第1、8、9、10章。东华大学许武军、白恩健、陈雯、蒋学芹等老师在编写过程中对教材的规划和编写给予了指导意见，提供了部分教材资料，并在C语言程序设计教学中提出了宝

贵的意见。东北大学龚涛、曾献辉、齐金鹏等老师对教材课件的编写提供了建议，并在教学中对该教材的使用提供了宝贵的意见。东华大学研究生陈建军、周凡、盛佐等完成了部分录入和校对工作。本书在编写过程中得到了东华大学信息学院领导、教师、学生的关心和支持，在此一并表示谢意。

由于作者水平有限，书中难免会存在缺点和不妥之处，敬请读者批评指正，请将意见和建议告诉我们，邮箱为 dhu.c.language@gmail.com。为了方便对本书内容进行交流和讨论，特提供一个 QQ 群：102084643。

作　者

2015 年 7 月

目　录

第 1 章　计算机程序设计概述

1.1　计算机系统组成

计算机是 20 世纪人类最伟大的创造发明之一，它是电子技术和计算技术空前发展的产物，是科学技术与生产力发展的结晶。计算机极大地推动了科学技术的发展，现已成为当今社会各行各业不可缺少的工具。

计算机是一种能够按照事先存储的程序，自动、高速地进行数值计算和信息处理的现代化智能电子设备。计算机系统由硬件系统和软件系统两部分组成。硬件系统是计算机系统的物理部分，是由电子线路、元器件和机械部件等构成的具体装置；软件系统是计算机系统中运行的各种程序、程序所需数据的集合。计算机系统的基本组成如图 1-1 所示。

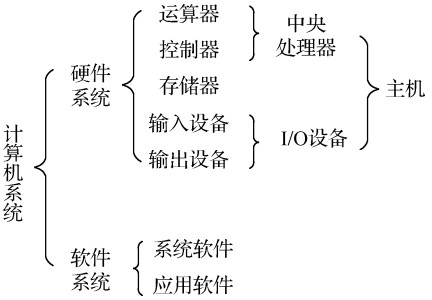


图 1-1　计算机系统的基本组成

通常将运算器和控制器称为中央处理器（Central Processor Unit，CPU），将输入设备和输出设备称为 I/O 设备（输入/输出，Input/Output），将中央处理器和 I/O 设备合称为主机。

1.1.1　硬件系统

当前计算机都是冯・诺依曼结构的，硬件系统由 5 部分构成，如图 1-2 所示，各部分功能如下。

（1）控制器。控制器为计算机的控制中心，从存储器中读取并分析指令，根据指令要求完成相应操作。控制器产生一系列控制命令，使硬件系统各部分协调工作，完成程序和数据的输入和运算并输出结果。

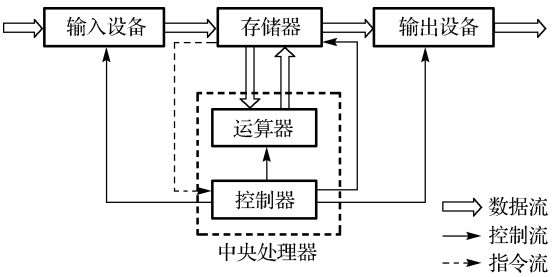


图 1-2　冯・诺依曼结构计算机

（2）运算器。运算器是中央处理器的执行单元，是所有中央处理器的核心组成部分。运算器在控制器的控制下，接收运算数据，完成指令指定的二进制算术运算。

（3）存储器。存储器是可以被中央处理器直接访问而无须通过输入/输出设备的记忆设备。存储器用来保存程序和数据，以及存储运算时的数据和结果。

（4）输入设备。输入设备是用来完成输入功能的部件。通过输入设备可以向计算机送入程序、数据及各种信息。常用的输入设备有键盘、鼠标、扫描仪、磁盘驱动器和触摸屏等。

（5）输出设备。输出设备是用来将计算机的中间运行情况或运行后的结果进行表现的设备。常用的输出设备有显示器、打印机、绘图仪和磁盘驱动器等。

根据冯·诺依曼结构，计算机自动执行程序，即执行指令，可分为如下几个过程。

（1）取指阶段。从存储器某地址处取出要执行的指令，送到中央处理器内部的指令寄存器中暂存。

（2）译码阶段。对保存在指令寄存器中的指令进行分析，翻译出该指令对应的操作。

（3）执行阶段。根据译码结果向各个部件发出相应控制信号，完成指令规定的操作。

（4）取下一条指令。为执行下一条指令做好准备，即产生下一条指令地址。

1.1.2 软件系统

软件系统为指挥计算机工作的程序和程序运行时所需要的数据，以及所需要的数据和文档的集合。软件系统分为系统软件和应用软件两部分。

（1）系统软件。计算机系统必备的基本软件，是管理、监控和维护计算机硬件和软件资源、开发应用的软件。按功能可分为 5 类：操作系统（如 Windows、Linux）、语言处理程序（如汇编程序、编译程序）、程序设计语言程序（如 C、Java 语言）、系统支持和服务程序（如系统诊断程序、查杀病毒程序）、数据库管理系统（如 Oracle、SQL Server）。

（2）应用软件。由系统软件开发的，为解决计算机各类应用问题而编写的程序，具有较强的实用性。按功能可分为两类：用户程序（如天气预报软件）和应用软件包（如 Microsoft Office 套件）。

1.2 程序设计语言

使用计算机，就需要和计算机交换信息。为解决人们和计算机交流的问题，就产生了程序设计语言。程序设计语言是用于编写计算机程序的语言，根据程序设计语言的发展，可分为三个阶段：很难理解的机器语言、较难理解的汇编语言、脱离机器的高级语言。

1．机器语言

机器语言是用二进制代码表示的计算机能直接识别和执行的一种机器指令的集合，是计算机唯一能够直接识别的程序设计语言。机器语言具有灵活、直接执行和速度快等特点。机器语言是直接对计算机硬件产生作用的，因此不同类型的计算机采用的机器语言是不同的，没有通用性，使得机器语言很难被人们掌握和推广，通常只有计算机专家或专业人员才使用机器语言。

【例 1-1】 以下是某 CPU 利用机器语言计算 a=a+b 的代码，其中 a=1，b=2。

```
#01: 11100011 10100000 00000000 00000001
#02: 11100011 10100000 00010000 00000010
#03: 11100000 10000000 00000000 00000001
```

代码解释：

#01：令 a=1；

#02：令 b=2；

#03：将 a 和 b 的值相加，并将结果放在 a 中。

2．汇编语言

为了克服机器语言难理解、难编写、难记忆和易出错的缺点，人们就用与代码指令意义相近的英文缩写词、字母和数字等符号来取代指令代码，于是产生了汇编语言。汇编语言采用了助记符来代表低级机器语言的操作，它和机器语言基本上是一一对应的，更便于记忆。

用汇编语言编写的程序称为汇编语言源程序，需要汇编程序将源程序汇编或翻译成机器语言源程序后，计算机才能执行。

汇编语言和机器语言都是面向机器的程序设计语言，不同的机器具有不同的指令系统，一般将它们称为低级语言。

【例 1-2】 以下是某 CPU 利用汇编语言计算 a=a+b 的代码，其中 a=1，b=2。

```
#01: MOV R0, #1
#02: MOV R1, #2
#03: ADD R0, R0, R1
```

代码解释：

#01：将数值 1 放入寄存器 R0 中；

#02：将数值 2 放入寄存器 R1 中；

#03：将寄存器 R0 的值和寄存器 R1 的值相加，结果放入寄存器 R0 中。

3．高级语言

计算机的发展促使人们去寻求一些与人类自然语言相接近且能为计算机所接受的语意确定、规则明确、自然直观和通用易学的计算机语言。这种与自然语言相近、与具体的计算机指令系统无关、其表达方式更接近人们对求解问题的描述方式的计算机语言称为高级语言。目前广泛使用的高级语言有 PASCAL、C、C++、Java、C#等。

使用高级语言编写的程序称为源程序，计算机并不能直接接受和执行用高级语言编写的源程序，需要通过翻译程序翻译成机器语言形式的目标程序，再与有关的库程序连接成可执行程序，计算机才能识别和执行。这种翻译通常有两种方式：编译方式和解释方式。

【例 1-3】 下面是利用 C 语言计算 a=a+b 的代码，其中 a=1，b=2。

```
#01: int a=1;
#02: int b=2;
#03: a=a+b;
```

代码解释：

#01：使变量 a 等于 1；

#02：使变量 b 等于 2；

#03：将变量 a 和变量 b 相加，结果赋值给变量 a。

1.3　计算机算法简介

做任何事情都有一定的步骤和方法，算法是在有限步骤内解决某一问题所使用的一组定义明确的规则。计算机算法是计算机能够执行的指令和规则，可分为两大类：数值运算算法和非数值运算算法。

1969 年，Edsger W. Dijkstra 首次提出了“结构化程序设计”的概念，1971 年，Niklaus E. Wirth 提出了“通过逐步求精方式开发程序”的思想，提出公式“算法+数据结构=程序”，一个程序应包括对数据的描述，在程序中要指定数据的类型和数据的组织形式，即数据结构（data structure）；和对操作的描述，即操作步骤，也就是算法（algorithm）。

Donald E. Knuth 在他的著作《计算机程序设计艺术》中将算法的特征归纳为如下特点。

（1）输入：一个算法有零个或多个输入。

（2）输出：一个算法应有一个或多个输出，输出是算法计算的结果。

（3）确定性：算法的描述必须无歧义，每个步骤应当是确定的，而不能是含糊的、模棱两可的，以保证算法的实际执行结果是准确符合要求或期望的，通常要求实际运行结果是确定的。

（4）有限性：一个算法应包含有限的操作步骤，而不能是无限的。

（5）有效性：又称可行性，算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

1.3.1 算法举例

对于计算机程序设计人员，必须会设计算法，并根据算法编写程序。本节给出一些简单问题的算法。

【例 1-4】 求解 5!=1×2×3×4×5。

解答：

（1）对于该问题，最原始方法按照如下步骤求解。

步骤 1：先求 1×2，得到结果 2；

步骤 2：将步骤 1 得到的乘积 2 乘以 3，得到结果 6；

步骤 3：将步骤 2 得到的乘积 6 再乘以 4，得到结果 24；

步骤 4：将步骤 3 得到的乘积 24 再乘以 5，得到结果 120。

这样的算法虽然正确，但太烦琐，不适合编写计算机程序。

（2）改进后的算法如下。

设变量 p 为被乘数，变量 i 为乘数。

S1：令 p=1；

S2：令 i=2；

S3：使 p×i，乘积仍然放在变量 p 中，可表示为 p×i→p；

S4：使 i 的值加 1，即 i+1→i；

S5：如果 i≤5，返回重新执行步骤 S3 以及其后的 S4 和 S5；否则，算法结束。

该算法不仅正确，而且是较好的计算机算法，因为计算机是高速运算的自动机器，实现循环（S3、S4 和 S5）非常容易。

（3）对于改进后的算法，很容易进行扩展。

如果计算 100!，只需将 S5 中的“i≤5”改成“i≤100”即可。

如果计算 1×3×5×7×9，算法只需改动如下。

S2：令 i=3；

S4：i+2→i；

S5：若 i≤11，返回 S3，否则结束。

【例 1-5】 求解 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$。

解答：

（1）最原始方法按照如下步骤求解。

步骤 1：求 $1-\frac{1}{2}$，得到 $\frac{1}{2}$。

步骤 2：将 $\frac{1}{2}+\frac{1}{3}$，得到 $\frac{5}{6}$。

步骤 3：将 $\frac{5}{6}-\frac{1}{4}$，得到 $\frac{7}{12}$。

……

这样的算法不适合编写计算机程序。

（2）改进后的算法如下。

设变量 sign 为运算符号（+或者−），sum 为运算结果，i 为序列 2, 3, 4,…, 100 中的一项：

S1：sign=1；

S2：sum=1；

S3：i=2；

S4：sign=(−1)×sign；

S5：item=sign×(1/i)；

S6：item =sum+item；

S7：i=i+1；

S8：若 i≤100，返回 S4；否则结束。

1.3.2　算法的表示方法

为了表示一个算法，可以采用不同的方法。常用的方法有自然语言、流程图、N-S 流程图、伪代码、计算机语言等。

1．用自然语言表示算法

自然语言就是人们常用的语言，如汉语或英语。用自然语言表示通俗易懂，但文字冗长，容易出现歧义，而且不方便用自然语言描述包含分支和循环的算法。除了很简单的问题，一般不用自然语言表示算法。

【例 1-6】 用自然语言描述、计算并输出 $y=\sqrt{x}$ 的算法。

解答：自然语言描述如下。

（1）输入 x。

（2）判断 x 是否小于零，如果成立，则输出错误信息；否则，计算 x 的平方根并赋值给 y，输出 y 的值。

2．用流程图表示算法

流程图是用一些约定的几何图形来描述算法，用流程图表示算法直观形象，易于理解。流程图的符号与意义如图 1-3 所示。

一个流程图包括表示相应操作的框、带箭头的流程线、框内外必要的文字说明等部分。

【例 1-7】 将求 5!= 1×2×3×4×5 的算法用流程图表示。

解答：用流程图表示求 5!的算法如图 1-4 所示。

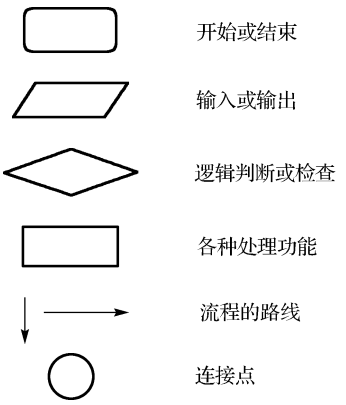


图 1-3　流程图的符号与意义

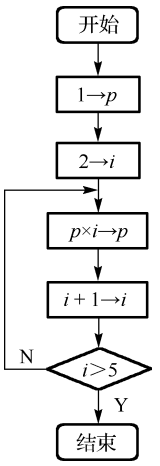


图 1-4　求 5!的算法的流程图表示

3．用 N-S 流程图表示算法

美国学者 I. Nasii 和 B. Shneiderman 于 1973 年提出了一种新的流程图形式，将全部算法写在一个矩形框内，完全去掉了带箭头的流程线，这种流程图称为 N-S 流程图。

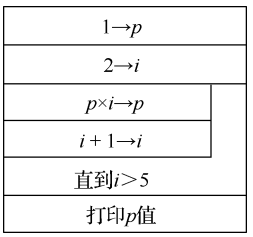


图 1-5　求 5!的算法的 N-S 流程图表示

【例 1-8】 将求 5!= 1×2×3×4×5 的算法用 N-S 流程图表示。

解答：用 N-S 流程图表示求 5!的算法如图 1-5 所示。

4．用伪代码表示算法

伪代码使用介于自然语言和计算机语言之间的文字和符号来描述算法，具有自然语言灵活的特点，同时又接近于程序设计语言的描述。

【例 1-9】 将求 5!= 1×2×3×4×5 的算法用伪代码表示。

解答：用伪代码表示求 5!的算法如下所示。

开始

　　置 *p* 的初值为 1

　　置 *i* 的初值为 2

　　当 *i*<=5，执行下面的操作：

　　　　使 *p*=*p*×*i*

　　　　使 *i*=*i*+1

　　　　(循环体到此结束)

　　打印 *p* 的值

结束

5．用计算机语言表示算法

我们的任务是用计算机解决问题，即用计算机实现算法，用计算机语言表示算法必须严格遵循所用语言的语法规则。

【例 1-10】 将求 5!= 1×2×3×4×5 的算法用计算机语言表示。

解答：用计算机语言表示求 5!的算法如下所示。

```
int product=1;
int i=2;

while (i<=5) {
    product=product*i;
    i=i+1;
}

printf("%d",product);
```

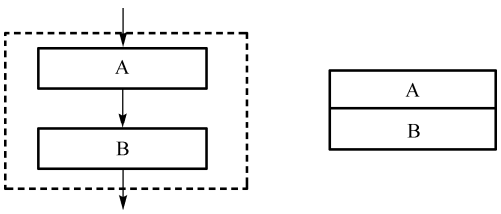
1.3.3　基本程序结构和流程图

已经证明：任何复杂的问题都可以用三种基本结构组成的程序完成。这三种基本结构是：①顺序结构，按指令的顺序依次执行；②选择结构，根据判别条件有选择地改变执行流程；③循环结构，有条件地重复地执行某个程序块。

三种基本结构有如下共同特点：①只有一个入口和出口；②结构内部的每部分都有机会被执行到；③结构内不存在“死循环”。

1．顺序结构

依次顺序地执行程序语句，如图 1-6 所示。

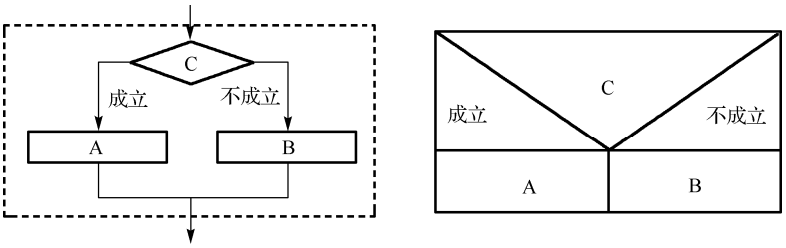


(a) 流程图　　(b) N-S流程图

图 1-6　顺序结构

2．选择结构

首先判断条件 C，若条件 C 成立，执行 A 程序块，否则执行 B 程序块，如图 1-7 所示。



(a) 流程图　　(b) N-S流程图

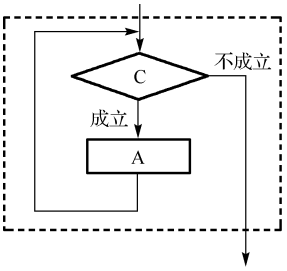
图 1-7　选择结构

3．循环结构

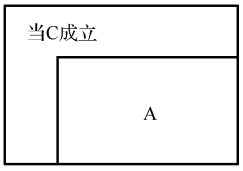
循环结构可以减少源程序书写的工作量，用来描述重复执行的某段程序块。循环结构分为当型循环和直到型循环。

（1）当型循环结构：当条件 C 成立时，就执行 A 程序块，然后再次判断条件 C 是否成立，直到条件 C 的值不成立才向下执行，如图 1-8 所示。

（2）直到型循环结构：先执行 A 程序块，然后判断条件 C 是否成立，若条件 C 不成立 ，再次执行 A 程序块，直到条件 C 成立时才向下执行，如图 1-9 所示。

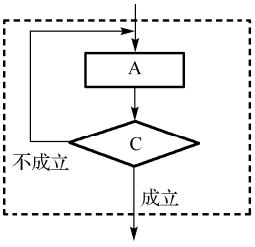


(a) 流程图

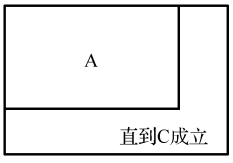


(b) N-S流程图

图 1-8　当型循环结构



(a) 流程图



(b) N-S流程图

图 1-9　直到型循环结构

1.4　数制及进制转换

数可以用两种方法来表示：①按“值”表示。在选定的进位制中表示出该数对应的值，称为进位制数；②按“形”表示。按照一定的编码方法，表示出该数特定的形式，称为编码制数。数的这两种表示方法涉及数制与编码，本节讲述数制及进制转换，下一节讲述数值编码。

1.4.1 基本进位制

1．十进制

十进制是最常用的数制，其特点如下。

（1）数码，表示数的符号。十进制数用 0、1、2、3、4、5、6、7、8、9 等 10 个符号表示，遵循“逢十进一”的原则，即 9+1=10。

（2）基数，为数码的个数。十进制数的基数是 10，即采用 10 个基本数码，任何一个十进制数都可以用 10 个数码按一定规律排列起来表示。

（3）位权，每一位所具有的值。0～9 这 10 个数可以用一位数码表示，10 以上的数则要用两位以上的数码表示。如数 12，右边的“2”为个位数，左边的“1”为十位数，也就是$12=1\times10^1+2\times10^0$。因此，每一数码处于不同位置时，它代表的数值是不同的，即不同的数位有不同的位权。

十进制数表示的数值等于其各位加权系数之和，如数 9876 可写为

$$9876=9\times10^3+8\times10^2+7\times10^1+6\times10^0$$

其中，每位的位权分别为 10^3、10^2、10^1、10^0。

一般地，任意一个 n 位十进制正整数 $(D)_{10}$ 均可表示为：

$$(D)_{10}=k_{n-1}\times10^{n-1}+k_{n-2}\times10^{n-2}+\cdots+k_1\times10^1+k_0\times10^0=\sum_{i=0}^{n-1}k_i\times10^i$$

其中，下脚注 10 表示括号内的数是一个十进制数，也可以用下脚注 D（Decimal）表示。k_i 是第 i 位的系数，为 0～9 当中的某一个数。如果一个数的整数部分有 n 位，小数部分有 m 位，则 i 的取值为 $-m$ 到 $n-1$ 之间（含 $-m$ 和 $n-1$）的所有整数。

【例 1-11】 十进制数 76543 和 2015.21 可表示如下：

$$(76543)_{\mathrm{D}}=(76543)_{10}=7\times10^4+6\times10^3+5\times10^2+4\times10^1+3\times10^0$$

$$(2015.21)_{\mathrm{D}}=(2015.21)_{10}=2\times10^3+0\times10^2+1\times10^1+5\times10^0+2\times10^{-1}+1\times10^{-2}$$

2．二进制

二进制是计算机中广泛采用的一种数制，其特点如下。

（1）数码。二进制数用 0 和 1 两个符号表示，遵循“逢二进一”的原则，即 1+1=10。

（2）基数。二进制数的基数是 2，即采用两个基本数码，任何一个二进制数都可以用两个数码按一定规律排列起来表示。

（3）位权。二进制的各位位权分别为 2^0、2^1、2^2、2^3……任意一个 n 位二进制正整数可表示为：

$$(D)_2=k_{n-1}\times2^{n-1}+k_{n-2}\times2^{n-2}+\cdots+k_1\times2^1+k_0\times2^0=\sum_{i=0}^{n-1}k_i\times2^i$$

其中，下脚注 2 表示括号内的数是一个二进制数，也可以用下脚注 B（Binary）表示。

二进制数表示的数值等于其各位加权系数之和，例如：

$$(1011)_{\mathrm{B}}=(1011)_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0=(11)_{10}$$

二进制数只有两个数码 0 和 1，可以用任何具有两个不同稳定状态的元件来表示，如晶体管的饱和与截止，继电器接点的闭合与断开等。只要规定其中一种状态表示 1，则另一种状态就表示 0，这样就可以表示二进制数了。因此，二进制的实现装置简单可靠，容易用二极管、晶体管等电子器件来实现，具有十进制无法具备的优点，因此广泛应用在计算机系统中。

二进制数的位数通常很多，不便于书写和记忆，例如，要表示十进制数 1111，若用二进制数表示则为 10001010111，若用八进制数表示则为 2127，若用十六进制数表示则为 457，因此在计算机程序设计中，常使用八进制数或十六进制数来表示二进制数。

3．十六进制

中国历史上曾在质量单位上使用过十六进制，如 16 两为一斤（1 斤=500g）。现在十六进制普遍应用在计算机领域。十六进制数特点如下。

（1）数码。十六进制数用 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 等 16 个符号表示（其中数值 10～15 分别用 A～F 来表示），遵循“逢十六进一”的原则，即 F+1=10。

（2）基数。十六进制数的基数是 16，即采用 16 个基本数码，任何一个十六进制数都可以用 16 个数码按一定规律排列起来表示。

（3）位权。十六进制的各位位权分别为 16^0、16^1、16^2、16^3……任意一个 n 位十六进制正整数可表示为：

$$(D)_{16}=k_{n-1}\times16^{n-1}+k_{n-2}\times16^{n-2}+\cdots+k_1\times16^1+k_0\times16^0=\sum_{i=0}^{n-1}k_i\times16^i$$

其中，下脚注 16 表示括号内的数是一个十六进制数，也可以用下脚注 H（Hexadecimal）表示。

十六进制数表示的数值等于其各位加权系数之和，例如：

$$(\mathrm{FE98})_{\mathrm{H}}=(\mathrm{FE98})_{16}=15\times16^3+14\times16^2+9\times16^1+8\times16^0=(65176)_{10}$$

4．八进制

在计算机中，八进制数有时可以取代十六进制数，如 Linux 系统的文件权限设置。十六进制数除了 0～9 之外，要用到 A～F 等符号，八进制数不必用数字以外的符号来表示。

八进制的特点如下。

（1）数码。八进制数用 0、1、2、3、4、5、6、7 等 8 个符号表示，遵循“逢八进一”的原则，即 7+1=10。

（2）基数。八进制数的基数是 8，即采用 8 个基本数码，任何一个八进制数都可以用 8 个数码按一定规律排列起来表示。

（3）位权。八进制的各位位权分别为 8^0、8^1、8^2、8^3……任意一个 n 位八进制正整数可表示为：

$$(D)_8=k_{n-1}\times8^{n-1}+k_{n-2}\times8^{n-2}+\cdots+k_1\times8^1+k_0\times8^0=\sum_{i=0}^{n-1}k_i\times8^i$$

其中，下脚注 8 表示括号内的数是一个八进制数，也可以用下脚注 O（Octal）表示。

八进制数表示的数值等于其各位加权系数之和，例如：

$$(7650)_{\mathrm{O}}=(7650)_8=7\times8^3+6\times8^2+5\times8^1+0\times8^0=(4008)_{10}$$

5．R 进制

综上所述，对一个 R（$R\geqslant2$）进制数，基数为 R，可以用 R 个符号表示一个 R 进制数，遵循“逢 R 进一”的原则。R 进制的各位位权分别为 R^0、R^1、R^2、R^3……任意一个 n 位 R 进制正整数可表示为：

$$(D)_{\mathrm{R}}=k_{n-1}\times R^{n-1}+k_{n-2}\times R^{n-2}+\cdots+k_1\times R^1+k_0\times R^0=\sum_{i=0}^{n-1}k_i\times R^i$$

其中，下脚注 R 表示括号内的数是一个 R 进制数，k_i 表示第 i 位的系数，R^i 为第 i 位的位权。R 进制数表示的数值等于其各位加权系数之和。

1.4.2 进制数间相互转换

1．二进制数、八进制数、十六进制数转换成十进制数

二进制数、八进制数、十六进制数转换成十进制数的方法是按权相加，即求出各位加权系数之和，得到相应的十进制数。

【例 1-12】 二进制数、八进制数、十六进制数转换成十进制数示例。

$$(111011)_2 = 1\times2^5 + 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = (59)_{10}$$

$$(F2E.A8)_{16} = 15\times16^2 + 2\times16^1 + 14\times16^0 + 10\times16^{-1} + 8\times16^{-2} = (3886.65625)_{10}$$

$$(1234)_8 = 1\times8^3 + 2\times8^2 + 3\times8^1 + 4\times8^0 = (668)_{10}$$

2．十进制数转换成二进制数、八进制数、十六进制数

将十进制数的整数部分转换为二进制数、八进制数、十六进制数时，可以采用“除 R 倒取余数”法，R 代表所要转换成的数制的基数，步骤如下。

步骤 1：把该十进制数的整数部分除以基数 R，取余数，即为最低位的数码 k_0。

步骤 2：将前一步得到的商再除以基数 R，再取余数，即得次低位的数码 k_1。

步骤 3：重复以上过程，直到商为 0 结束，最后得到的余数即为最高位数的数码 k_{n-1}。

将十进制数小数部分转换为二进制数、八进制数、十六进制数时，可以采用“乘 R 顺取整数”法，R 代表所要转换成的数制的基数，步骤如下。

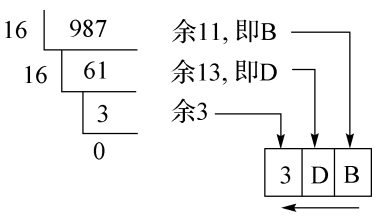
步骤 1：把该十进制数的小数部分乘以基数 R，取整数，即得小数的最高位数码 k_{-1}。

步骤 2：将前一步得到的乘积的小数部分再乘以基数 R，再取整数，即得小数的次低位数码 k_{-2}。

步骤 3：重复以上过程，直到乘积为零，或者达到所需求的精度为止。最后一次取的整数为小数的最低位 k_{-m}。

【例 1-13】 把十进制数 987 转换成十六进制数。

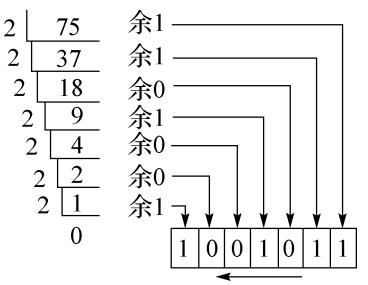
解答：



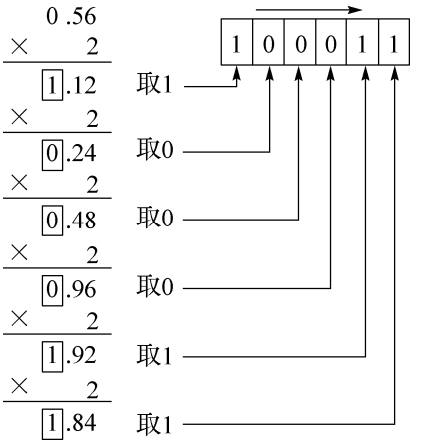
因此，$(987)_{10} = (3DB)_{16}$。

【例 1-14】 将十进制数 75.56 转换为二进制数（误差 $\varepsilon < 1/2^6$）。

解答：（1）整数部分，采用“除 2 倒取余数”法转换。



（2）小数部分，采用“乘 2 顺取整数”法转换。



因此，$(75.56)_{10}=(1001011.100011)_2$ 转换到第 6 位小数时误差 $\varepsilon<1/2^6$ 。

3．二进制数与十六进制数的相互转换

（1）将二进制正整数转换为十六进制数

将二进制数从右向左开始，每 4 位分为一组（不足 4 位左补 0），每组都相应转换为一位十六进制数。

【例 1-15】 将二进制数 11010110111101 转换为十六进制数。

解答：

二进制　0011　0101　1011　1101

十六进制　3　5　B　D

因此，$(110101101111101)_2=(35BD)_{16}$ 。

（2）将十六进制正整数转换为二进制数

将十六进制数的每一位转换为相应的 4 位二进制数即可。

【例 1-16】 将十六进制数 F9E8 转换为二进制数。

解答：

十六进制　F　9　E　8

二进制　1111　1001　1110　1000

因此，$(F9E8)_{16}=(1111100111101000)_2$。

4．二进制数与八进制数的相互转换

（1）将二进制正整数转换为八进制数

将二进制数从右向左开始，每 3 位分为一组（不足 3 位左补 0），每组都相应转换为一位八进制数。

【例 1-17】 将二进制数 11010110111101 转换为八进制数。

解答：

二进制　011　010　110　111　101

八进制　3　2　6　7　5

因此，$(11010110111101)_2=(32675)_8$。

（2）将八进制正整数转换为二进制数

将八进制数的每一位转换为相应的 3 位二进制数即可。

【例 1-18】 将八进制数 7654 转换为二进制数。

解答：

八进制　7　6　5　4

　　↓　↓　↓　↓

二进制　111 110 101 100

因此，$(7564)_8=(111110101100)_2$。

为了方便转换，表 1-1 分别给出了二进制数/十六进制数和二进制数/八进制数的换算表。

表 1-1　二进制数/十六进制数和二进制数/八进制数的换算表

二 进 制 数	十六进制数	二 进 制 数	十六进制数	二 进 制 数	八 进 制 数
0000	0	1000	8	000	0
0001	1	1001	9	001	1
0010	2	1010	A	010	2
0011	3	1011	B	011	3
0100	4	1100	C	100	4
0101	5	1101	D	101	5
0110	6	1110	E	110	6
0111	7	1111	F	111	7

1.5 数 值 编 码

编码是信息从一种形式或格式转换为另一种形式或格式的过程。常用的编码有两类：一类是十进制编码，如美国信息交换标准代码（ASCII）；另一类是二进制编码，如数的机器码表示。

1.5.1 美国信息交换标准代码（ASCII）

ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）是基于拉丁字母的一套计算机编码系统，主要用于显示现代英语。ASCII 是现今最通用的单字节编码系统，被国际标准化组织（ISO）选定为一种国际通用的代码，广泛用于通信和计算机中。

标准 ASCII 码也称为基础 ASCII 码，使用 7 位二进制数来表示所有的大写和小写字母、数字 0～9、标点符号及一些特殊控制字符。例如，空格的 ASCII 码为二进制数 0100000；字符 0 的 ASCII 码为二进制数 0110000；字符 a 的 ASCII 码为二进制数 1100001；字符 A 的 ASCII 码为二进制数 1000001；控制符换行的 ASCII 码为二进制数 0001010；控制符回车的 ASCII 码为二进制数 0001101。

附录 B.1 给出了 ASCII 代码的对照表。

1.5.2 数的机器码表示

在计算机中为了表示正、负数，在数的最高位前设置一个符号位，并规定符号位为“0”时表示该数为正数；符号位为“1”时表示该数为负数。这种带有符号位的数称为机器数，机器数有原码、反码、补码三种表示形式。

1．原码

二进制原码最高位为符号位，其余各位为数值本身的绝对值，又称为“符号+绝对值”表示法。符号位“0”表示正数，符号位“1”表示负数。

例如，$(+99)_{10}$的带符号位 8 位原码表示为$(01100011)_{原}$，其中最高位的 0 代表正数符号的符号位，后 7 位是代表 99 这个数的二进制表示法。$(-99)_{10}$的原码可表示为$(11100011)_{原}$，其中最高位的 1 代表负数符号的符号位，后 7 位代表–99 这个数的绝对值的二进制表示法。

零的原码有两种表示形式：$(+0)_{原}$ = 00000000，$(-0)_{原}$ = 10000000，机器遇到这两种情况都当作 0 处理。

原码表示法简单易懂，但在进行加、减法运算时，符号位不能直接参加运算，而是要分别计算符号位和数值位。当两数相加时，如果是同号，则数值相加；如果是异号，则要进行减法运算，此时还要比较两数的绝对值大小，先用大数减去小数，最后还要判断符号位，这样会导致运算速度将低。

为了解决该问题，引入数的反码和补码表示法。

2．反码

引入反码是便于求负数的补码。二进制反码表示法的规则是：正数的反码与原码相同，负数的反码是符号位为 1，数值是对应原码各位取反。

例如，$(+99)_{10}$ 的带符号位 8 位反码表示为$(01100011)_{反}$，与原码相同。$(-99)_{10}$ 的反码可表示为$(10011100)_{反}$，其中最高位的 1 代表负数，后 7 位是–99 的原码各位取反。

零的反码有两种表示形式：$(+0)_{反}=00000000$，$(-0)_{反}=11111111$。

3．补码

二进制补码表示法的规则是：正数的补码与原码相同，负数的补码是符号位为 1，数值位逐位取反（求其反码），然后在最低位对整个数加 1。

例如，$(+99)_{10}$ 的带符号位 8 位补码表示为$(01100011)_{补}$，与原码相同。$(-99)_{10}$ 的补码可表示为$(10011101)_{补}$，其中最高位的 1 代表负数，后 7 位是–99 的原码的数值部分各位取反后加 1。

零的补码只有一种形式：$(+0)_{补}=(-0)_{补}=00000000$。

一个负数的二进制补码转换成十进制数的规则是：最高位不变，其余位取反后加 1 得到原码。例如，补码 11111001，取反后为 10000110（最高位不变），然后加 1 得 10000111，即十进制数–7。

采用补码表示法进行二进制数的加、减法运算，符号位可以和数值位一起参与运算，减法运算可转换为加法运算，得到的运算结果是补码的形式。两个补码的符号位和数值部分产生的进位相加得到的和，就是运算结果的符号。在用补码计算时，要注意补码的位数必须足够多，能表示运算的绝对值，否则会得到错误的运算结果。

【例 1-19】 若 $a=13$，$b=21$，计算 $a-b$ 的值。

解答：用补码的计算方法为：$a-b=a+(-b)=(a)_{补}+(-b)_{补}=(13)_{补}+(-21)_{补}$

因为$(a)_{补}=00001101$，$(-21)_{补}=11101011$，则 $a-b=11111000$。

结果为补码，转换为原码为 10001000，所以 $a-b$ 的结果为十进制数–8。

表 1-2 给出了几个数的原码、反码、补码的对照表（用 8 位表示）。

表 1-2　原码、反码、补码对照表

十 进 制 数	二 进 制 数		
	原　码	反　码	补　码
+0	00000000	00000000	00000000
–0	10000000	11111111	00000000
+1	00000001	00000001	00000001
–1	10000001	11111110	11111111

（续表）

十 进 制 数	二 进 制 数		
	原　码	反　码	补　码
+7	00000111	00000111	00000111
–7	10000111	11111000	11111001
数值范围	11111111～01111111 (–127～–0, +0～+127)	10000000～01111111 (–127～–0, +0～+127)	10000000～01111111 (–128～0～+127)

1.6 C 语言概述

1.6.1 C 语言简介

C 语言是一种通用的编程语言，广泛用于系统软件与应用软件的开发。C 语言具有高效、灵活、功能丰富、表达力强和可移植性好等特点，在程序员中备受青睐，成为近几十年来使用最广泛的编程语言之一。目前，C 语言编译器普遍存在于各种不同的操作系统中，如 Microsoft Windows、Mac OS X、Linux、UNIX 等。C 语言的设计影响了众多后来的编程语言，如 C++、Objective-C、Java、C#等。

1．发展过程

C 语言的发展过程大致可分为三个阶段：Old Style C、C89 和 C99。Old Style C 指 Ken Thompson 和 Dennis Ritchie 最初发明 C 语言的标准。C 语言源于 BCPL 语言，后者由 Martin Richards 于 1967 年左右所设计。1970 年，Ken Thompson 为运行在 PDP-7 上的首个 UNIX 系统设计了一个精简版的 BCPL 语言，被称为 B 语言。在 PDP-11 计算机出现后，Dennis Ritchie 与 Ken Thompson 着手将 UNIX 系统移植到 PDP-11 上，由于 B 语言自身的特点不适合，为此 Dennis Ritchie 与 Ken Thompson 以 B 语言为基础，在贝尔实验室设计和开发出来。

1973 年，UNIX 操作系统内核正式用 C 语言改写，这是 C 语言第一次应用在操作系统的开发上。1975 年 C 语言开始移植到其他计算机中，Stephen C. Johnson 实现了一套“可移植编译器”，从此，C 语言在大多数计算机上被使用，从最小的微型计算机到 CRAY-2 超级计算机。1978 年，Dennis Ritchie 和 Ken Thompson 合作出版了《C 程序设计语言》，书中介绍的 C 语言标准也被 C 语言程序员称为“K&R C”。

1989 年，C 语言被美国国家标准协会（ANSI）标准化，扩展 K&R C，增加了一些新特性，被称为 C89，是最早的 C 语言规范。

1990 年，国际标准化组织（ISO）成立工作组规定国际标准的 C 语言，通过对 ANSI 标准的少量修改，最终制定了 ISO 9899:1990，又称为 C90。随后，ANSI 亦接受国际标准 C 语言。

C89 是目前最广泛采用的 C 语言标准，大多数编译器都完全支持 C89。经典的 C 语言教材《C 程序设计语言》（第二版）就是基于这个版本的。

在 ANSI 的标准确立后，C 语言的规范在一段时间内没有大的变动。《标准修正案一》在 1994 年为 C 语言创建了一个新标准，支持更多的国际字符集，这个标准被称为 C99（ISO 9899:1999）。C99 被 ANSI 于 2000 年 3 月采用，但目前仍没有得到广泛支持。

2011 年，ISO 正式发布了 C 语言的新标准 C11，之前被称为 C1X，官方名称为 ISO/IEC 9899:2011。

2．C 语言特点

由于 C 语言的强大功能和各方面的优点，在各类大中小和微型计算机上得到了广泛的使用，成为现在最优秀的程序设计语言之一。C 语言有如下特点。

（1）C 语言简洁，紧凑，使用方便，灵活。C 语言语法限制不太严格，程序设计自由度较大。

（2）运算符丰富。C 语言共有 34 种，把括号、赋值、逗号等都作为运算符处理，从而使运算类型极为丰富，可以实现其他高级语言难以实现的运算。

（3）数据结构类型丰富。

（4）具有结构化的控制语句。

（5）C 语言允许直接访问物理地址，能进行位（bit）操作，能实现汇编语言的大部分功能，可以直接对硬件进行操作。

（6）用 C 语言写的程序可移植性较好，生成目标代码质量高，程序执行效率高。

1.6.2　C 语言程序示例

以下通过两个例子来说明 C 语言源程序的基本部分和结构特点。

【例 1-20】 程序 1-1：在屏幕上输出“hello, world!”字符串，通常是初学编程语言时的第一个简单程序。

```
#01: //程序 1-1
#02: #include <stdio.h>
#03: int main(void){
#04:    printf("hello, world!\n");
#05:    return 0;
#06: }
```

程序解释：

#01：程序注释，以“//”开始的行为程序注释，不产生编译代码。

#02：编译预处理指令，包含标准输入/输出头文件，用来提供输入/输出功能。

#03：主函数的函数定义。每个 C 语言源程序都必须有且只能有一个 main 函数。

#04：函数调用语句，printf( )函数是由系统定义的标准函数，可在程序中直接调用，功能是把要输出的内容输出到屏幕显示。

#05：程序运行结束。

程序运行结果如下：

```
hello, world!
```

【例 1-21】 程序 1-2：输入一个实数值，求出正弦值后在屏幕上输出。

```
#01: //程序 1-2
#02: #include <stdio.h>
#03: #include <math.h>
#04: int main(){
#05:    double x = 0.0;
#06:    double s = 0.0;
#07:
#08:    printf("Input a number:");
#09:    scanf("%lf",&x);
#10:    s = sin(x);
#11:    printf("sin(%lf)=%lf\n",x,s);
```

```
#12:
#13:     return 0;
#14: }
```

程序解释：

#03：math.h 为数学运算函数的头文件，用 include 指令包含该头文件，以在程序中使用正弦函数。

#05, #06：定义两个实数变量 x 和 s，程序后面将使用。

#08：显示提示信息，提示用户输入一个数值。

#09：从键盘获得一个实数，保存在 x 中。

#10：求 x 的正弦，结果保存在变量 s 中。

#11：显示程序的运算结果。

程序运行结果如下（下画线表示用户输入内容）：

```
Input a number:10
sin(10.000000)=-0.544021
```

被#include 指令包含的文件通常是由系统提供的，其扩展名为“.h”，因此也称为头文件。C 语言的头文件中包括各个标准库函数的函数原型，在程序中调用一个库函数时，必须包含该函数原型所在的头文件。

1.6.3 C 语言程序编译与执行

对于高级语言程序，需要使用编译器/解释器将其转换成机器代码后才能在计算机中运行。解释器像一位“中间人”，每次执行程序时都要先转换成另一种语言再执行，因此解释器的程序运行速度比较缓慢。常见的解释程序有 BASIC、python、LISP 等。相对地，编译器一次将程序翻译成机器码，可以多次执行而无须再编译，其生成程序无须依赖编译器就可执行，程序运行速度比较快。常见的编译程序有 C、C++、Pascal 等。

如图 1-10 所示，对于 C 语言程序，从编辑源代码到程序运行需要 4 个步骤。

（1）编辑代码。输入编辑程序源代码，生成源程序或头文件。源程序是用户创建的文件，以“.c”为文件扩展名保存。头文件是含有函数的声明和预处理语句，用于帮助访问外部定义的函数，扩展名为“.h”。

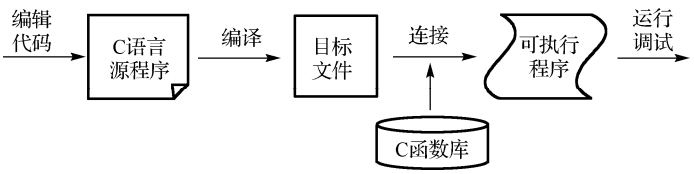


图 1-10 C 语言程序编译与执行

（2）编译。对语法进行分析，翻译生成目标程序。目标文件是编译器的输出结果，常见扩展名为“.o”或“.obj”。

（3）连接。与其他目标程序或库连接装配，生成可执行程序。可执行程序是连接器的输出结果，常见扩展名为“.exe”。

（4）运行，对程序运行调试，输出结果。

通常把程序的编译和连接统称为编译阶段。

上机实验：熟悉 C 语言开发环境

本次实验熟悉 C 语言的开发环境，通过运行简单的 C 语言程序，初步了解 C 语言源程序的特点。了解在 Windows 系统上如何编辑、编译和运行一个 C 语言程序。

Microsoft Visual C++（简称 VC++）是微软公司的集成开发工具，可提供 C 语言、C++及 C++/CLI 等编程语言。VC++提供了一个集源程序编辑、代码编译与调试于一体的开发环境，称为集成开发环境，能够提高软件开发效率。程序员通过 VC++可以访问 C 语言源代码编辑器、资源编辑器，使用内部调试器等。VC++被整合在 Visual Studio 之中，但仍可单独安装使用。

下面使用 VC++编写并运行一个简单的程序。

（1）运行 VC++，选择“文件”菜单中的“新建”命令，打开“新建”对话框，选择“文件”选项卡，在左边的列表框中选择“C++ Source File”项，在右边的“文件名”编辑框中输入文件名“addtest.c”，文件名可以任意取，但是要求文件扩展名为“.c”，以说明是一个 C 语言程序。可以选择合适的位置保存该源文件，如图 1-11 所示。

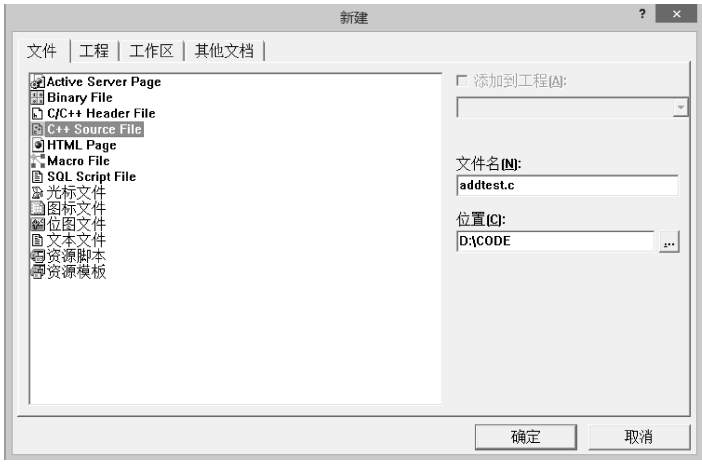


图 1-11　“新建”对话框设置

（2）在窗口中编辑如下的源程序。

```
//calculate the sum of a and b
#include <stdio.h>
int add(int,int);
int main(){
    int a,b,sum;
    a=1;
    b=2;
    sum=add(a,b);
    printf("%d + %d = %d\n",a,b,sum);
    return 0;
    }
```

```
        int add(int x,int y){
        return(x+y);
    }
```

（3）选择“组建”菜单中的“组建”命令，对程序进行编译连接，在编译过程中，VC++会生成一个同名的工作区，不用理会。

如果程序没有错误，则在下方的信息窗口中显示：“-0 error(s)，0 warning(s)”，得到可执行程序addtest.exe。

有时会出现一些警告性信息（warning），不影响程序的执行。如果程序中有 N 个致命性错误（error），则会提示：“-N error(s)，M warning(s)”，此时，拖动信息窗口右侧的滚动条，可以看到所有编译器给出的错误提示，包括错误在第几行，以及是什么错误。双击某行出错信息，程序窗口中会用箭头指示对应出错位置。根据信息窗口中的错误提示修改错误，修改好后，保存并重新编译，直到无错误为止。

（4）选择“组建”菜单中的“执行”命令，运行程序，观察结果。

（5）完成后，选择“文件”菜单中的“关闭工作区”命令，关闭程序工作区。

习　题

1．用流程图表示给出求闰年问题的算法。

判断 1～2100 年中的每一年是否为闰年，将结果输出。

闰年的条件：（1）能被 4 整除，但不能被 100 整除的年份；（2）能被 100 整除，又能被 400 整除的年份。

2．用流程图表示给出求素数问题的算法。

对一个大于或等于 2 的正整数，判断它是不是一个素数。

素数的条件：除了 1 和它本身外，不能被其他自然数整除，也称质数。

3．用流程图表示求解如下问题的算法。

（1）输入三个数 a、b、c，分别输出最大的数和最小的数。

（2）求 $1+2+3+\cdots+99+100$ 的和。

（3）求一元二次方程 $ax^2+bx+c=0$ 的根。注意考虑 $\Delta=b^2-4ac$ 大于、等于、小于零的情况。

4．将下列二进制数、八进制数和十六进制数转换为十进制数。

二进制数：（1）1111，（2）10101010；

八进制数：（1）1111，（2）76543210；

十六进制数：（1）1111，（2）FFFF，（3）FEDCBA。

5．将下列十进制数分别转换为二进制数、八进制数和十六进制数。

（1）1234；（2）9999；（3）255；（4）10.25。

6．将下列二进制数分别转换为八进制数和十六进制数。

（1）1111；（2）10101010；（3）1111111；（4）11111111。

7．将下列八进制数和十六进制数转换为二进制数。

八进制数：（1）1111，（2）76543210；

十六进制数：（1）1111，（2）FFFF，（3）FEDCBA。

8．求出下列十进制数的原码、反码和补码（用 8 位二进制数表示）。

（1）1；（2）–1；（3）63；（4）–63；（5）127；（6）–127。

第 2 章　C 语言基础

2.1　基 本 知 识

2.1.1　位和字节

位、字节和字的概念如下。

（1）位

位（bit）是计算机中最基本的单位，每一位的状态只能是 0 或 1。在计算机中信息的表示方式为一串逻辑 0 和逻辑 1 组成的二进制字码，例如，10110111，其中每个逻辑 0 或 1 便是一位，10110111 共有 8 位。

（2）字节

8 个二进制位构成一字节（Byte），它是存储空间的基本计量单位。一字节可以存储一个英文字母，一个汉字占据 2 字节的存储空间。例如，字符 A 用 0100 0001 表示。

对于 KB，K 表示 1024，即 2^{10}。1KB 表示 2^{10} 个 Byte，即 1024 字节。

对于 MB，1MB = 2^{20} Byte = 1024×1024 Byte=1024 KB

对于 GB，1GB=1024 MB= 1024×1024 KB = 20^{30} Byte

（3）字

字（word）是计算机进行数据处理和运算的单位。字由若干字节构成，字的位数称为字长，不同类型计算机有不同的字长。例如，一台 8 位机，它的一个字就等于 1 字节，字长为 8 位。如果是一台 32 位机，它的 1 个字就由 4 字节构成，字长为 32 位。

2.1.2　标识符

在 C 语言程序中，标识符是用来标识变量、常量、函数名等的字符序列。C 语言规定，标识符只能是由字母（A～Z, a～z）、数字（0～9）、下画线（_）组成的字符串，并且其第一个字符必须是字母或下画线，而且标识符区分大小写，不能与关键字相同。

例如，这些标识符是合法的：x，abc，name，stu123，sum，Sum，day，Date，_above 等。

这些标识符是不合法的：

3com，-ax，7days，#33，$123 等，原因是第一个字符不符合要求。

a*b，boy-girl，M.D.John，a>b 等，原因是出现非法字符。

char，goto，for 等，原因是与 C 语言关键字重复。

使用标识符时应注意如下事项。

（1）C 语言标准不限制标识符的长度，但标识符长度受不同 C 语言编译器或不同类型计算机限制。例如，某 C 语言编译器中规定标识符为 8 位，则当两个标识符前 8 位相同时，被认为是同一个标识符。使用时一般不要超过 32 个字符。

（2）标识符区分大小写，如 BOOK 和 book 是两个不同的标识符。

（3）标识符虽然可以随意定义，但标识符是用于标识符某个量的符号，因此，命名应尽量有相应的意义，以便于阅读理解。

2.1.3 数据类型

C 语言程序要求使用的各种变量都要预先定义，即先定义后使用。变量定义包括三个方面：数据类型、存储类型、作用域。在此只介绍数据类型，其他之后再介绍。

在 C 语言中，按照被定义变量的性质、表示形式、占据存储空间的大小等特点，数据类型可分为 4 类：基本类型、构造类型、指针类型、空类型，如图 2-1 所示。

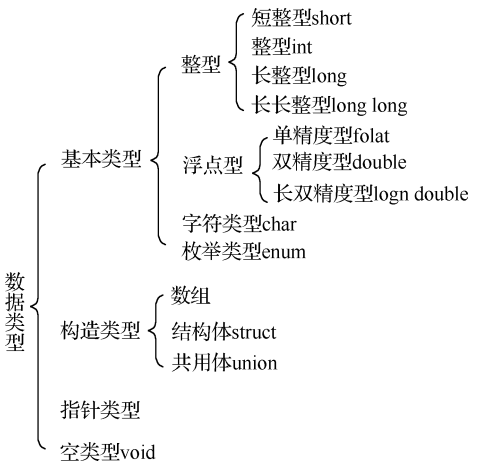


图 2-1 数据类型分类

（1）基本类型

基本类型的值不可以再分解为其他类型，即基本类型是自我说明的。

基本类型量按取值是否可改变，可分为常量和变量两类。在程序执行过程中，其值不发生改变的量称为常量，其值可变的量称为变量。常量是可以不经说明而直接引用的，而变量则必须先定义后使用。它们可与数据类型结合起来进行分类，如整型常量、整型变量、浮点常量、浮点变量、字符常量、字符变量、枚举常量、枚举变量等。

基本类型主要有整型、实型（浮点型）、字符型、枚举类型，其中，整型又分为短整型、整型、长整型、长长整型等，实型又分为单精度型、双精度型和长双精度型等。

表 2-1 所示为 32 位计算机中典型的基本类型数据位长和取值范围，不同编译器可能使用不同的数据位长和范围。

表 2-1 基本类型数据位长和取值范围

类 型	符 号	关 键 字	字 节 数	取 值 范 围
整型	有	(signed) short (int)	2	$-32768\sim32767$，即$-2^{15}\sim2^{15}-1$
		(signed) int	4	$-2^{31}\sim2^{31}-1$
		(signed) long (int)	4	$-2^{31}\sim2^{31}-1$
		(signed) long long (int)	8	$-2^{63}\sim2^{63}-1$
	无	unsigned short (int)	2	$0\sim65535$，即$0\sim2^{16}-1$
		unsigned int	4	$0\sim2^{32}-1$
		unsigned long (int)	4	$0\sim2^{32}-1$
		unsigned long long (int)	8	$0\sim2^{64}-1$
实型	有	float	4	$\pm1.2\times10^{-38}\sim\pm3.4\times10^{38}$，0（6 位有效数字）
	有	double	8	$\pm2.3\times10^{-308}\sim\pm1.7\times10^{308}$，0（15 位有效数字）
	有	long double	8	$\pm2.3\times10^{-308}\sim\pm1.7\times10^{308}$，0（15 位有效数字）
字符型	有	(signed) char	1	$-128\sim127$，即$-2^{7}\sim2^{7}-1$
	无	unsigned char	1	$0\sim255$，即$0\sim2^{8}-1$

（2）构造类型

构造类型是根据已定义的一个或多个基本类型用构造的方法来定义的，也就是说，一个构造类型的值可以分解成若干“成员”或“元素”，每个“成员”都是一个基本类型或又是一个构造类型。

构造类型主要有数组、结构体、共用体（或联合体）等。

（3）指针类型

指针是一种特殊的、很重要的数据类型，用来表示某个变量在内存储器中的地址。指针变量的取值类似于整型量，但这是两个类型完全不同的量。

（4）空类型

在调用函数时，通常应向调用者返回一个函数值，该返回值具有一定的数据类型，应在函数定义时说明。但有时，函数调用后并不需要向调用者返回函数值，这种函数可以定义为“空类型”，其类型说明符为 void。

本章主要介绍基本类型中的整型、浮点型和字符型，其余类型在以后章节中介绍。

2.2　常　　量

在程序执行过程中其值不发生改变的量称为常量，可分为符号常量和直接常量两类。

1．符号常量

在 C 语言中，可以用一个标识符来表示一个常量，称之为符号常量。符号常量在使用之前必须先定义，其一般形式为：

```
#define 标识符 常量
```

其中，#define 也是一条预处理指令，称为宏定义，其功能是把该标识符定义为其后的常量值。定义后，在程序中所有出现该符号常量的地方，均可用该常量值替代。

【例 2-1】 程序 2-1：符号常量（PRICE）的使用。

```
#01: //程序 2-1
#02: #define PRICE 30
#03:
#04: int main(){
#05:     int num=10;
#06:     int total;
#07:
#08:     total=num*PRICE;
#09:     printf("total=%d\n",total);
#10:
#11:     return 0;
#12: }
```

程序解释：

#02：用标识符 PRICE 代表一个常量 30，称为符号常量 PRICE。

#08：此处 PRICE 被 30 替代，即等价于：total=num*30。

程序运行结果如下：

```
total=300
```

使用符号常量的好处是：含义清楚，能做到“一改全改”。

符号常量与变量不同，它的值在其作用域内不能改变，也不能再被赋值。通常为了区分变量和符号常量，变量用小写字母，符号常量用大写字母。

2．直接常量

直接常量，即字面常量，又可分为整型常量（如 1，–1）、实型常量（如 3.14，–1.5）、字符常量（如'a'，'A'）、字符串常量（如"hello"，"China"）等。

2.2.1 整型常量

整型常量就是整常数。在 C 语言中，使用的整常数有八进制、十六进制和十进制三种。

（1）十进制整常数

十进制整常数没有前缀，由数字 0～9 和正、负号表示。如 123、–128、65535、–1 等都是合法的十进制整常数。非法的十进制整常数如 010（不能有前缀 0）和 98A（不能含有非十进制数码）。

C 语言程序是根据前缀来区分八进制数和十六进制数的，因此书写常数时，不要把前缀弄错而造成结果不正确。

（2）八进制整常数

八进制整常数必须以 0 开头，即以 0 作为八进制数的前缀，数码取值为 0～7。八进制数通常是无符号数。如 011、0123、0177777 等都是合法的八进制整常数。非法的八进制整常数如 255（无前缀 0）、01FF（包含了非八进制数码）、–0123（出现了负号）。

（3）十六进制整常数

十六进制整常数的前缀为 0X 或 0x，其数码取值为 0～9、A～F 或 a～f。例如 0x123、0xFFFF、0XAA 等都是合法的十六进制整常数。非法的十六进制整常数如 9A（无前缀 0x）、0x12H（含有非十六进制数码）。

（4）整型常量类型

在整型常量后加后缀“L”或“l”，则称为长整型常数（long int）。

整型常量根据其值所在范围确定其数据类型。在 32 位计算机中，基本整型的长度为 32 位，表示的数的范围是有限定的，如果超过了范围，此时就必须用长整型常数（long int）来表示。

长整型常数如：

十进制长整型常数：888L、32767L、8589934592L；

八进制长整型常数：012L、0777L、010000000L；

十六进制长整型常数：0xFFL、0x1234L、0x10000L。

长整型常数在运算和输出格式上要予以注意，避免出错。例如，长整型常数 888L 和基本整型常数 888 在数值上并无区别，但对 888L，因为是长整型常量，编译器将为它分配 8 字节的存储空间；而对 888，因为是基本整型，只分配 4 字节的存储空间。

可用后缀“U”或“u”表示整型常数的无符号数。如 123U、0xFFU、888LU 等均为无符号数。

前缀和后缀可同时使用，以表示各种类型的整型数，如 0x123FEDLU 表示十六进制无符号长整型数 123FED。

2.2.2 实型常量

实型也称为浮点型，实型常量也称为实数或浮点数。在 C 语言中，实数只采用十进制形式，有两种表示形式：十进制小数形式和指数形式。

（1）十进制小数形式

十进制小数形式由正负号、数码 0～9 和小数点组成，必须包含小数点。如 0.0、3.14、–0.15、0.123、.123、123.、123.0 等都是合法的十进制数形式。非法的小数形式如：123（无小数点）、9A.E（非十进制数码）。

（2）指数形式

数学中可以用幂的形式来表示实数，如 12.34 可以表示为 1.234×10^{1}，指数形式与之类似，由正负号、十进制数、阶码标志“e”或“E”及阶码组成，“e”或“E”之前必须有数字，指数必须为整数，“e”或“E”的前、后及数字之间不能有空格。如 $12.3e^{3}$、$123E^{2}$、$1.23e^{-1}$ 等都是合法的指数形式，非

法的指数形式如 e^{-5}（阶码标志 E 之前无数字），$1.2E^{2.5}$（阶码非整数），53.–E3（负号位置不对）、.7E（无阶码）等。

指数形式的一般形式为：aE^n 或 ae^n，其中，a 为十进制数，n 为十进制整数，其值为 $a\times10^n$。例如，$6.02E^{23}$ 等价于 6.02×10^{23}。

C 标准允许浮点数使用后缀。没有后缀的浮点型常量默认情况下为 double 型，在浮点型常量后加字母 f 或 F，则认为它是 float 型。

在十进制整型常数后面加后缀“f”或“F”即表示该数为浮点数，如 123f 和 123. 是等价的。

2.2.3　字符常量

字符常量是用单引号括起来的单个普通字符或转义字符，如'a'、'1'、'='、'\n'、'\101'等都是合法的字符常量。

1．普通字符常量

普通字符常量有以下特点：

（1）只能用单引号括起来，不能用双引号或其他括号；

（2）只能是单个字符，不能是字符串；

（3）字符常量的值是该字符的 ASCII 码值，如 'a' 等于 97，'A' 等于 65。

（4）字符可以是字符集中的任意字符，但数字被定义为字符型之后就不代表数值的含义，如'1'和 1 是不同的，'1'是字符常量，值等于 49，而 1 是数值 1。

2．转义字符

转义字符是一种特殊的字符常量，以反斜线“\”开始，后面跟一个字符或一个代码值。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。例如，在例 2-1 中的 printf( )函数中用到的“\n”就是一个转义字符，其意义是“回车换行”。转义字符主要用来表示那些用一般字符不便于表示的控制代码。

常用转义字符及其含义如表 2-2 所示。

表 2-2　常用转义字符及其含义

转义字符	含　义	转义字符	含　义
\n	换行	\t	水平制表
\v	垂直制表	\b	退格
\r	回车	\f	换页
\a	响铃	\\	反斜线
\’	单引号	\"	双引号
\ddd	三位八进制数代表的字符	\xhh	两位十六进制数代表的字符

C 语言字符集中的任何一个字符均可用转义字符来表示，表 2-2 中的“\ddd”和“\xhh”正是为此而提出的，其中 ddd 和 hh 分别为八进制数和十六进制数对应的 ASCII 代码，如 '\101' 表示字母 A，'\102' 表示字母 B，'\134' 表示反斜线，'\X0A' 表示换行等。

【例 2-2】 程序 2-2：转义字符的使用。

```
//程序 2-2
#include <stdio.h>
```

```
int main(){
   printf("\101 \x42 C\n");
   printf("I say:\"How are you?\"\n");
   printf("\\C Program\\\n");
   printf("Language \'C\'");

   return 0;
}
```

程序运行结果如下：

```
A B C
I say:"How are you?"
\C Program\
Language 'C'
```

2.2.4 字符串常量

字符串常量是由一对双引号括起来的字符序列，每个字符串尾自动加一个空字符 '\0' 作为字符串结束标志，例如，"Shanghai"、"Language C"、"Hello world" 等都是合法的字符串变量。

字符串常量和字符常量不同，主要有以下区别。

（1）字符常量由单引号括起来，字符串常量由双引号括起来。

（2）字符常量只能是单个字符，字符串常量可以含一个或多个字符。

（3）C 语言中只有字符变量，没有字符串变量。可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。

（4）字符常量占一字节的内存空间，字符串常量占的内存字节数等于字符串中的字符数加 1。增加的一个字节中存放字符串结束标志 "\0"（ASCII 码为 0）。

例如，字符串“Hello world”在内存中存放情况为：

H	e	l	l	o		w	o	r	l	d	\0

字符常量 'a' 和字符串常量 "a" 虽然都只有一个字符，但在内存中的存放情况不同：

'a' 在内存中占 1 字节，可表示为：

a

"a" 在内存中占 2 字节，可表示为：

a	\0

2.3 变　　量

在程序执行过程中其值可以改变的量称为变量。一个变量应该有一个名字，在内存中占据一定的存储单元。

（1）变量定义

变量定义的一般格式为：

数据类型　变量 1[, 变量 2, …, 变量 n];

数据类型决定分配的字节数和数的表示范围，变量为合法标识符。

数据类型说明符与变量名之间至少用一个空格间隔，可以同时定义多个相同类型的变量，之间用“,”隔开。

（2）变量初始化

在定义变量时就赋初值，称为变量初始化。变量定义必须放在变量使用之前，一般放在函数体的开头部分。

在变量定义中赋初值的一般形式为：

类型说明符 变量 1=值 1, 变量 2=值 2, …;

初值可以是另外一个变量的值，例如：

```
int start=0;
int sum=start;
```

在定义中不允许连续赋值，如“int a=b=c=1;”是不合法的。

【例 2-3】 变量定义与变量初始化。

```
//变量定义
int sum;
long width, long;
float x, y, z;
double area;
//变量初始化
double r=1.5;
char ch='A';
```

基本数据类型变量可分为整型变量、实型变量、字符变量等。

2.3.1 整型变量

整型数据在内存中以补码的形式存放，整型变量可分为：短整型、整型、长整型、长长整型等，每种又可以分为无符号型和有符号型两种，各种类型的数据位长和取值范围见表 2-1，其中关键字列中“( )”的意义是可有可无。

（1）短整型：类型说明的关键字为 short int 或 short，在内存中占 2 字节。

（2）整型：类型说明的关键字为 int，在内存中占 4 字节。

（3）长整型：类型说明的关键字为 long int 或 long，在内存中占 4 字节。

（4）长长整型：类型说明的关键字为 long long int 或 long long，在内存中占 8 字节。

这些类型默认是有符号的，可表示正数和负数。如果要只表示正数，可以用无符号型，类型说明符为 unsigned，无符号型可与上述 4 种类型匹配而构成：无符号短整型（unsigned short 或 unsigned short int）、无符号整型（unsigned int）、无符号长整型（unsigned long 或 unsigned long int）、无符号长长整型（unsigned long long 或 unsigned long long int）等。各种无符号类型量所占的内存空间字节数与相应的有符号类型量相同，但不能表示负数。

C 语言没有明确定义整数类型的大小，不同整数类型的取值随机器的不同而不同，但规定整型类型不能比短整型类型短、长整型不能比整型类型短、长长整型不能比长整型类型短，所以各种类型所占的内存空间字节数满足：short≤int≤long≤long long。

【例 2-4】 程序 2-3：整型变量的定义与使用。

```
//程序 2-3
#include <stdio.h>
```

```
int main(){
    int a,b,c,d;
    unsigned x;

    a=10;
    b=-10;
    x=1;
    c=a+x;
    d=b+x;
    printf("a+x=%d,b+x=%d\n",c,d);

    return 0;
}
```

程序运行结果如下：

```
a+x=11,b+x=-9
```

在定义和使用整型变量时，注意各种类型的取值范围，以免发生数据溢出而造成错误。

【例 2-5】 程序 2-4：整型数据的溢出。

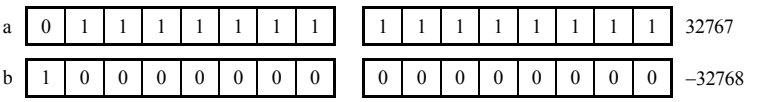
```
//程序 2-4
#include <stdio.h>
int main(){
    short int a,b;
    a=32767;
    b=a+1;
    printf("a=%d, b=%d\n",a,b);

    return 0;
}
```

程序运行结果如下：

```
a=32767, b=-32768
```

此例中 short int 类型在内存中占 2 字节，a、b 在内存中存放情况如下，a+1 后，符号位从 0 变为 1，结果成为负数，即–32768 的补码。



2.3.2 实型变量

目前 C 编译器都遵照 IEEE 制定的浮点数表示法进行实型数值运算，该标准采用一种科学计数法，用符号、指数和尾数来表示，底数为 2，即把一个浮点数表示为尾数乘以 2 的指数次方再添上符号。图 2-2 所示为浮点数表示法示意图，并给出了一个 float 类型变量 sum=8.5 在内存中的存放形式。

由于 $(8.5)_{10}=1.0625\times2^3=(1+0.0625)_{10}\times2^3=(1+0.0001)_2\times2^3$，即小数部分为二进制 0.0001，指

数部分为 3，所以在内存中存放时，符号位为 0（正数），尾数为 0.0001（小数部分的二进制值），阶码为 10000010（130 = 127+3 的二进制值，阶码等于指数加上 127），其中阶码的存储采用了移位存储方法，具体可查阅 IEEE 754 标准。

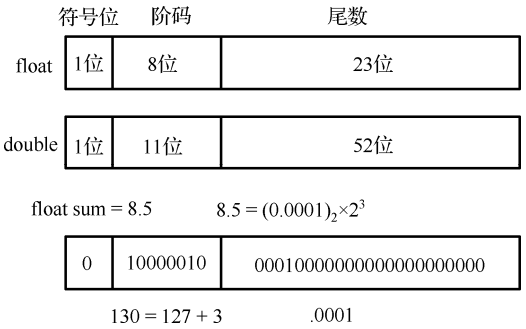


图 2-2　浮点数表示法

由图 2-2 可以看出，小数部分占的位数越多，数的有效数字越多，精度越高。相反地，指数部分占的位数越多，则能表示的数值范围越大。

实型变量可分为单精度型、双精度型和长双精度型三类，各种类型的数据位长和取值范围如表 2-1 所示。

（1）单精度型，类型说明的关键字为 float，在内存中占 4 字节，为 6 位有效数字。

（2）双精度型，类型说明的关键字为 double，在内存中占 8 字节，为 15 位有效数字。

（3）长双精度型，类型说明的关键字为 long double，在内存中占 8 字节，为 15 位有效数字。

实型变量定义格式和使用方法与整型变量相同，如“float x,y;”。

由于实型变量是由有限的存储单元组成的，因此能提供的有效数字总是有限的，可能存在舍入误差。

【例 2-6】 程序 2-5：实型变量的舍入误差。

```
//程序 2-5
#include <stdio.h>
int main(){
    float a;
    double b;

    a=33333.33333;
    b=33333.333333333333333;
    printf("a=%f\nb=%.15lf\n",a,b);

    return 0;
}
```

程序运行结果如下：

```
a=33333.332031
b=33333.333333333336000
```

由于 a 是单精度浮点型，有效位数为 6 位，整数部分已占 5 位，故小数 1 位后之后均为无效数字。b 是双精度型浮点型，有效位为 16 位，整数部分已占 5 位，故小数 11 位后之后均为无效数字。

2.3.3 字符变量

字符变量用来存储单个字符的 ASCII 值，字符变量的类型说明符是 char。

字符变量类型定义格式和使用与整型变量相同，如“char ch;”。

每个字符变量在内存中被分配一字节的内存空间，因此只能存放一个字符，存储的字符值是对应的 ASCII 码。例如，字符 'A' 的十进制 ASCII 码是 65，对字符变量 ch 赋予 'A' 值：“ch='A';”，则在 ch 存储单元内存放 65 的二进制代码。

字符变量与整型变量间可进行算术运算，C 语言允许对整型变量赋以字符量，也允许对字符变量赋以整型值。在输出时，允许把字符变量按整型变量输出，也允许把整型变量按字符变量输出。字符变量为单字节量，整型变量为多字节量，当整型变量按字符量处理时，只有低 8 字节参与处理。

【例 2-7】 程序 2-6：字符变量与整型变量之间的运算。

```
#01: //程序 2-6
#02: #include <stdio.h>
#03: int main(){
#04:     char a,b,x,y;
#05:
#06:     a=65;
#07:     b=66;
#08:     printf("a=%c,b=%c\n",a,b);
#09:     printf("a=%d,b=%d\n",a,b);
#10:
#11:     x='a';
#12:     y='b';
#13:     x=x-32;
#14:     y=y-32;
#15:     printf("x=%c,%d,y=%c,%d\n",x,x,y,y);
#16:     return 0;
#17: }
```

程序解释：

#04：定义 a、b、x、y 为字符型变量。

#06, #07：在赋值语句中赋予字符变量 a、b 整型值。

#08：a、b 值的输出形式取决于 printf( )函数格式串中的格式符，当格式符为“c”时，输出变量值为字符。

#09：当格式符为“d”时，输出变量值为整数。

#11, #12：x、y 被赋予字符值。

#13, #14：允许字符变量参与数值运算，即用字符的 ASCII 码参与运算。大、小写字母的 ASCII 码相差 32，因此运算后把小写字母换成大写字母。

#15：分别以整型和字符型输出。

程序运行结果如下：

```
a=A,b=B
a=65,b=66
x=A,65,y=B,66
```

2.4　数据类型转换

整型、浮点型、字符型数据间可以混合运算，即变量的数据类型是可以转换的，如 10+'a'+1.5–8765.1234*'b' 是合法的。转换方法有两种：隐式转换（或自动转换）和显式转换（或强制转换）。

1．隐式转换

隐式转换发生在不同数据类型的量混合运算时，由编译器自动完成，下列情况会发生隐式转换：

（1）运算转换，不同类型数据混合运算时；

（2）赋值转换，把一个值赋给与其类型不同的变量时；

（3）输出转换，输出时转换成指定的输出格式；

（4）函数调用转换，实参与形参类型不一致时转换。

对于运算转换，其规则为：

（1)不同类型数据运算时先自动转换成同一类型，然后进行运算；

（2）转换按类型长度增加的方向进行，以保证精度不降低，如 int 型和 long 型运算时，先把 int 量换转成 long 型后再进行运算；

（3）所有的浮点型运算都是以双精度进行的，即使仅含 float 单精度量运算的表达式，也要先转换成 double 型，再进行运算；

（4）char 型和 short 型参与运算时，必须先转换成 int 型。

图 2-3 所示为类型自动转换的规则。

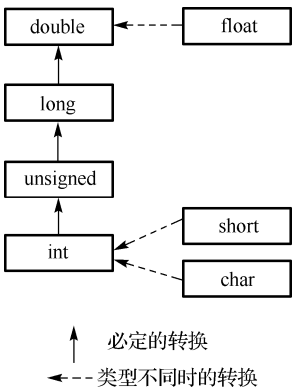
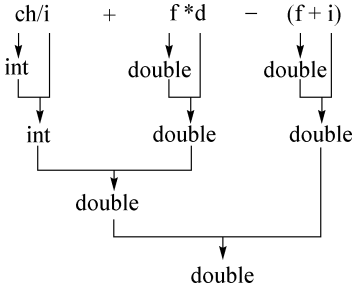


图 2-3　类型自动转换的规则

例如，定义变量如下：

```
char ch;
int i;
float f;
double d;
```

如果进行运算 ch/i + f *d–(f+i)，则类型自动转换方式为：



对于赋值运算，其规则为：当赋值号两边量的数据类型不同时，赋值号右边量的类型将转换为左边量的类型。如果右边量的数据类型长度大于左边时，将丢失一部分数据，这样会降低精度，丢失的部分按四舍五入舍入。

【例 2-8】 程序 2-7：类型隐式转换。

```
#01: //程序 2-7
#02: #define PI 3.1416
```

```
#03: #include <stdio.h>
#04: int main(){
#05:     int girth=0,area=0;
#06:     int radius=10;
#07:
#08:     girth=2*PI*radius;
#09:     area=PI*radius*radius;
#10:     printf("girth=%d,area=%d\n",girth,area);
#11:
#12:     return 0;
#13: }
```

程序解释：

#02：PI 为实型常量，定义为圆周率。

#05：girth、area 为整型。

#08：radius 转换成 double 计算，结果为 double 类型。但由于 girth 为整型，故赋值结果仍为整型，舍去小数部分。

#09：radius 转换成 double 计算，结果为 double 类型。但由于 area 为整型，故赋值结果仍为整型，舍去小数部分。

#10：输出结果为整数。

程序运行结果如下：

```
girth=62,area=314
```

2．显式转换

显式转换是通过类型转换运算来实现的，一般形式为：

（类型说明符）（表达式）

其功能是把表达式的运算结果强制转换成类型说明符所表示的类型。

例如，(int)(x+y) 的意义是把 x+y 的结果转换为整型；(int)x+y 的意义是把 x 转换为整型，然后与 y 相加。

显式转换的类型说明符和表达式都是必须加括号（单个变量可以不加括号），如(int)(x+y)和(int)x+y 的意义不同。

需要注意的是，无论是隐式转换还是显式转换，都只是为了本次运算的需要而对变量的数据类型进行的临时性转换，而不改变数据说明时对该变量定义的类型。

【例 2-9】 程序 2-8：类型显式转换。

```
#01: //程序 2-8
#02: #include <stdio.h>
#03: int main(){
#04:     float  x=3.6;
#05:     int  i;
#06:
#07:     i=(int)x;
#08:     printf("x=%f,i=%d\n",x,i);
```

```
#09:
#10:    return 0;
#11: }
```

程序解释：

#07：变量 x 强制转换为 int 型并赋值给变量 i，但只在运算中起作用，是临时的，而变量 x 本身的类型并不改变。

#08：输出时变量 i 的值为 3（删去了小数），x 的值仍为 3.6。

程序运行结果如下：

```
x=3.600000,i=3
```

2.5　运算符和表达式

C 语言中的运算符和表达式非常多，使得 C 语言功能很完善，C 语言的运算符具有不同的优先级和结合性。表达式中各运算量参与运算的先后顺序不仅要遵守运算符优先级的规定，还要受运算符结合性的制约，以便确定是自左向右进行运算，还是自右向左进行运算。

C 语言的运算符可分为以下几类。

（1）算术运算符，用于各类数值运算。例如，加（+）、减（–）、乘（*）、除（/）、求余（%）、自增（++）、自减（––）等。

（2）关系运算符，用于比较运算。例如，大于（>）、小于（<）、等于（=）、大于等于（>=）、小于等于（<=）和不等于（!=）等。

（3）逻辑运算符，用于逻辑运算。例如，与（&&）、或（||）、非（!）等。

（4）位操作运算符，按二进制位进行运算。例如，位与（&）、位或（|）、位非（～）、位异或（^）、左移（<<）、右移（>>）等。

（5） 赋值运算符，用于赋值运算。例如，简单赋值（=）、复合算术赋值（+=, –=, *=, /=, %=）、复合位运算赋值（&=, |=, >>=, <<=）等。

（6）条件运算符，是一个三目运算符，用于条件求值（?:）。

（7）逗号运算符，用于把若干表达式组合成一个表达式（,）。

（8）指针运算符，用于取内容（*）和取地址（&）运算。

（9）求字符数运算符，用于计算数据类型所占的字节数（sizeof）。

（10）特殊运算符，例如，括号（），下标[ ]，成员（–>，.）等。

本节只介绍算术运算符、赋值运算符和逗号运算符，其他运算符在后续章节介绍。

表达式是由常量、变量、函数和运算符等组合起来的式子，单个常量、变量、函数可以看作是表达式的特例。

一个表达式有一个值及其类型，它们等于计算表达式所得结果的值和类型，表达式求值按运算符的优先级和结合性规定的顺序进行。

C 语言中运算符的运算优先级共分为 15 级，1 级最高，15 级最低。在表达式中，优先级较高的先于优先级较低的进行运算，而在一个运算量两侧的运算符优先级相同时，则按运算符的结合性所规定的结合方向处理。

运算符的结合性分为两种：左结合性（自左向右）和右结合性（自右向左）。

各种运算符的运算优先级和结合性可查阅附录 B.2。

2.5.1 算术运算符和算术表达式

算术运算符用于各类数值运算，算术运算符有以下 7 种。

（1）加法运算符“+”。加法运算符为双目运算符，即有两个量参与加法运算，如 x+y、1+2 等，具有右结合性。

（2）减法运算符“–”。减法运算符为双目运算符，具有右结合性。

“+”和“–”可作为正值、负值运算符，此时为单目运算，具有左结合性。如+1、–2、–a 等。

（3）乘法运算符“*”。乘法运算符为双目运算符，具有左结合性。

（4）除法运算符“/”。除法运算符为双目运算符，具有左结合性。参与运算量均为整型时，结果也为整型，舍去小数。

“+、–、*、/”运算中，如果运算量中有一个是实型（float 或 double），则结果为双精度实型（double）。

（5）求余运算符“%”。求余运算符为双目运算符，具有左结合性。求余运算的结果等于两数相除后的余数，要求参与运算的量均为整型。

（6）自增运算符“++”。其功能是使变量的值增加 1。

（7）自减运算符“––”。其功能是使变量的值减小 1。

自增、自减运算符均为单目运算符，都具有右结合性，可有以下几种形式：

① ++i。i 自增 1 后再参与其他运算。

② ––i。i 自减 1 后再参与其他运算。

③ i++。i 参与运算后，i 的值再自增 1。

④ i––。i 参与运算后，i 的值再自减 1。

“++”和“––”不能用于常量和表达式，如 1++、(a+b)++等是非法的。

【例 2-10】 程序 2-9：算术运算符。

```
//程序 2-9
#include <stdio.h>
int main(){
   printf("%d,%d\n",8/3,-8/3);
   printf("%f,%f\n",8.0/3,-8.0/3);
   printf("%d\n",10%3);

   return 0;
}
```

程序运行结果如下：

```
2,-2
2.666667,-2.666667
1
```

【例 2-11】 程序 2-10：自增自减运算。

```
#01: //程序 2-10
#02: #include <stdio.h>
#03: int main(){
#04:     int x,y;
#05:     int a,b,c;
```

```
#06:
#07:    x=1; y=++x;
#08:    printf("x=%d,y=%d\n",x,y);
#09:    x=1; y=x++;
#10:    printf("x=%d,y=%d\n",x,y);
#11:    x=1; y=--x;
#12:    printf("x=%d,y=%d\n",x,y);
#13:    x=1; y=x--;
#14:    printf("x=%d,y=%d\n",x,y);
#15:
#16:    a=1;b=2;
#17:    c=(++a)*b;
#18:    printf("c=%d\n",c);
#19:    a=1;b=2;
#20:    c=(a++)*b;
#21:    printf("c=%d\n",c);
#22:
#23:    return 0;
#24: }
```

程序解释：

#07：变量 x（初始值为 1）先自增 1 后（值为 2），再赋值给变量 y（值为 2）。

#09：变量 x（初始值为 1）先赋值给变量 y（值为 1），再自增 1（值为 2）。

#11：变量 x（初始值为 1）先自减 1 后（值为 0），再赋值给变量 y（值为 0）。

#13：变量 x（初始值为 1）先赋值给变量 y（值为 1），再自减 1（值为 0）。

#17：变量 a（初始值为 1）先自增 1 后（值为 2），再参与运算 a*b，计算结果为 4。

#20：变量 a（初始值为 1）先参与运算 a*b，计算结果为 2，再自增 1（值为 2）。

程序运行结果如下：

```
x=2,y=2
x=2,y=1
x=0,y=0
x=0,y=1
c=4
c=2
```

算术表达式是由算术运算符和括号将运算对象（也称操作数）连接起来的、符合 C 语言语法规则的式子。如“3.14*r*r+2*3.14*r*h”、“(a+b)/(c–d)”等。

算术表达式运算规则：先按运算符的优先级别高低次序执行；若在运算对象两侧的运算符有相同的优先级，则按规定的结合方向顺序处理。

算术运算符的优先级如图 2-4 所示。

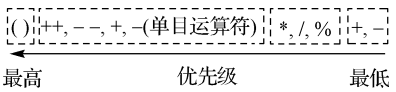


图 2-4　算术运算符的优先级

算术运算符的结合性是自左至右，即先左后右。例如，表达式“x–y+z”，则 y 应先与“–”号结合，执行 x–y 运算，然后再执行+z 的运算。这种自左至右的结合方向称为“左结合性”。而自右至左的结合方向称为“右结合性”，例如，表达式“–i++”，则先计算(i++)，然后取负值，即等价于“–(i++)”。

2.5.2 赋值运算符和赋值表达式

1．简单赋值运算符

简单赋值运算符为“=”，由“=”连接的式子称为赋值表达式，其一般形式为：

变量=表达式

其含义是将一个数据（常量或表达式）赋给一个变量，例如“x=a+b”、“area=2*3.14*radius”等都是合法的赋值表达式。

赋值表达式的功能是计算表达式的值再赋予左边的变量，赋值运算符具有右结合性，例如表达式“a=b=c=5”等价于“a=(b=(c=5))”。

凡是表达式可以出现的地方，均可以出现赋值表达式，如表达式“x=(a=1)+(b=2)”是合法的赋值表达式，其含义是把 1 赋予变量 a、2 赋予变量 b，再把变量 a、b 相加，和赋予变量 x，故变量 x 值等于 3。

赋值表达式左侧必须是变量，不能是常量或表达式，如“3=x–2*y”、“a+b=3”都是非法的赋值表达式。

2．复合赋值运算符

在赋值运算符“=”之前加上其他二目运算符，可构成复合赋值运算符，如+=、–=、*=、/=、%=等。

构成复合赋值表达式的一般形式为：

变量 双目运算符= 表达式

等价于：

变量=变量　运算符　表达式

例如，“a+=2”等价于“a=a+2”，“x*=y+z”等价于“x=x*(y+z)”。

3．赋值运算类型转换

如果赋值运算符两侧的数据类型不一致，将会自动进行类型转换，赋值转换的规则是：使赋值运算符右边表达式的值自动转换成赋值运算符左边变量的类型，具体如下：

（1）实型赋值给整型，则舍去实型的小数部分；

（2）整型赋值给实型，数值不变，但以浮点型存放，即增加小数部分（小数部分的值为 0）；

（3）无符号字符型赋值给整型，由于字符型为 1 字节，整型大于等于 2 字节，故把字符 ASCII 码值放入整型变量的低 8 位中，高位全部补 0；

（4）有符号字符型赋值给整型，则把字符 ASCII 码值放入整型变量的低 8 位中，其他位进行符号扩展：若字符最高位为 0，则整型变量其他位补 0；若字符最高位为 1，则整型变量其他位补 1；

（5）整型赋值给字符型，只把低 8 位赋值给字符量；

（6）无符号型赋值给有符号型，赋值运算符两边类型长度相等，但取值范围不同，右边的正数范围要比左边大一倍，因此只有在左边的正数范围内赋值才正确，否则所赋的正值在变量中会得到负的结果；

（7）有符号型赋值给无符号型，正数可以照原值传送，但若传送负数，则左边会得到一个正数，其原因在于把原来的符号位也当成了数值位。

【例 2-12】 程序 2-11：赋值运算类型转换。

```
#01: //程序 2-11
#02: #include <stdio.h>
```

```
#03: int main(){
#04:    unsigned short int ui;
#05:    short int i;
#06:    unsigned char ch1='\376';
#07:    char ch2='\376';
#08:
#09:    i=ch1;
#10:    printf("(int)=(unsigned char)\'\\376\', i=%d\n",i);
#11:    i=ch2;
#12:    printf("(int)=(signed char)\'\\376\', i=%d\n",i);
#13:
#14:    i=256;
#15:    ch2=i;
#16:    printf("(char)=(int)256, char=%d\n",ch2);
#17:
#18:    ui=65535;
#19:    i=ui;
#20:    printf("(int)=(unsigned int)65535, int=%d\n",i);
#21:
#22:    i=-1;
#23:    ui=i;
#24:    printf("(unsigned int)=(int)-1,unsigned int=%d\n",ui);
#25:
#26:    return 0;
#27: }
```

程序解释：

#10：无符号字符型赋值给整型，转换示意图如图 2-5(a)所示。

#12：有符号字符型赋值给整型，转换示意图如图 2-5(b)所示。

#16：整型赋值给字符型，转换示意图如图 2-5(c)所示。

#20：无符号型赋值给有符号型，转换示意图如图 2-5(d)所示。

#24：有符号型赋值给无符号型，转换示意图如图 2-5(e)所示。

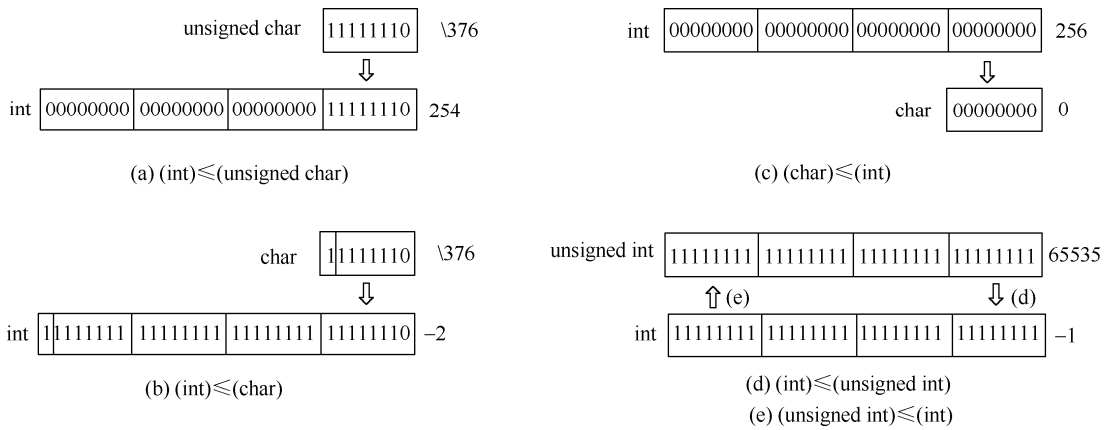


图 2-5　赋值运算类型转换示意图

程序运行结果如下：

```
(int)<=(unsigned char)'\376', i=254
(int)<=(char)'\376', i=-2
(char)<=(int)256, char=0
(int)<=(unsigned int)65535, int=-1
(unsigned int)<=(int)-1,unsigned int=65535
```

2.5.3 逗号运算符和逗号表达式

逗号运算符“,”的功能是把两个表达式连接起来组成一个表达式，称为逗号表达式。并不是在所有出现逗号的地方都组成逗号表达式，如在变量说明、函数参数表中，逗号只是用作各变量之间的间隔符。

逗号表达式一般形式为：

```
表达式 1，表达式 2，…，表达式 n
```

逗号表达式的求值过程是先求表达式 1 的值，再求表达式 2 的值，……，最后求表达式 n 的值，并将表达式 n 的值作为整个逗号表达式的值。

程序中使用逗号表达式，通常是要分别求逗号表达式内各表达式的值，并不一定求整个逗号表达式的值。

【例 2-13】 程序 2-12：逗号表达式。

```
//程序 2-12
#include <stdio.h>
int main(){
    int x,y=7;
    float z=4;

    x=(y=y+6,y/z);
    printf("x=%d\n",x);

    return 0;
}
```

程序运行结果如下：

```
x=3
```

2.5.4 C 语言语句

任何表达式在其末尾加上分号就构成为语句，如“x=a+b;”、“a=b=c=1;”等都是赋值语句。C 语言程序的主体部分是由语句组成的，以实现程序的功能。C 语言的语句可分为 5 类。

（1）表达式语句。表达式语句由表达式加上分号“;”组成，执行表达式语句就是计算表达式的值，一般形式为“表达式;”，如“a++;”、“x=(a+b)/(a–b) ;”等。

（2）函数调用语句。由函数名、实际参数加分号“;”组成，执行函数调用语句就是调用函数，执行被调函数体中的语句，一般形式为“函数名(实际参数表);”，如“printf(“Hello world!\n”);”。

（3）控制语句。由特定的语句定义符组成，用于控制程序的流程，以实现程序的各种结构，可分为三类：

① 条件判断语句，如 if 语句、switch 语句等；

② 循环执行语句，如 do while 语句、while 语句、for 语句等；

③ 转向语句，如 break 语句、goto 语句、continue 语句、return 语句等。

（4）复合语句。把多个语句用花括号“{ }”括起来组成的一个语句称复合语句。复合语句内各语句都必须以分号“;”结尾，在花括号“}”外不能加分号。例如，下列语句是一条复合语句。

```
{
    x=y-z;
    a=b-c;
    printf("x=%d, a=%d\n",x, a);
}
```

（5）空语句。只有分号“;”组成的语句称为空语句。空语句是什么也不执行的语句，在程序中，空语句可用作空循环体。例如，“while (getchar( )!='\n') ;”中，while 的循环体为空语句，完成功能：只要从键盘输入的字符不是回车则重新输入。

由赋值表达式加上分号构成的表达式语句称为赋值语句，其功能和特点与赋值表达式相同，一般形式为“变量=表达式;”。

赋值语句有如下特点：

（1）在赋值符“=”右边的表达式也可以是一个赋值表达式，如“变量=（变量=表达式);”是合法的。以此类推，一般形式为“变量=变量=…=表达式;”，如“a=b=c=d=1;”等价于 4 条语句：“d=1; c=d; b=c; a=b;”（注意赋值运算符具有右结合性）。

（2）在变量说明中给变量赋初值是变量说明的一部分，赋初值后的变量与其他同类变量之间仍用逗号间隔，如“int x=1,y,z;”。赋值语句必须以分号结尾。

（3）在变量说明中不允许连续给多个变量赋初值，如“int a=b=c=1;”是非法的。赋值语句允许连续对多个变量赋值。

上机实验：C 语言基础知识

本次实验的目的是掌握 C 语言的数据类型，学会使用 C 语言的有关算术运算符和表达式。

（1）输入下列程序，运行并分析结果。

```
#include<stdio.h>
int main(){
    char ch1, ch2;
    ch1=97;ch2=98;
    printf("ch1=%c,ch2=%c\n",ch1,ch2);
    ch1=300; ch2=400;
    printf("ch1=%c,ch2=%c\n",ch1,ch2);

    return 0;
}
```

（2）输入下列程序，运行并分析结果。

```
#include<stdio.h>
int main(){
```

```
    int i, j, m, n;
    i=8; j=10;
    m=++i;
    n=j++;
    printf("i=%d,j=%d,m=%d ,n=%d \n",i,j,m,n);
    return 0;
}
```

（3）输入下列程序，运行并分析结果。

```
#include<stdio.h>
int main(){
    short int x,y;
    x=32767;
    y=-32768;
    x=x+1 ;
    y=y-1;
    printf("x=%d,y=%d\n",x,y );
    return 0;
}
```

习　题

1．求出下列表达式的结果。

（1）10%3　　（2）10/3　　（3）int a=12; a+=a–=a*a

（4）int s=6，s%2+(s+1)%2　　（5）a=2,b=5,a++,b++,a+b

（6）ch='a'+'8'–'3'

2．定义“int b=7; float a=2.5,c=4.7;”，求表达式 a+(int)(b/3*(int)(a+c)/2)%4 的值。

3．定义“int a=2,b=3;float x=3.5,y=2.5;”，求表达式(float)(a+b)/2+(int)x%(int)y 的值。

4．定义“int e=1,f=4,g=2;float m=10.5,n=4.0,k;”，求赋值表达式 k=(e+f)/g+sqrt((double)n) *1.2/g+m 的值。

5．输出下列程序的运行结果。

```
#include<stdio.h>
int main(){
    int sum = 5;
    int sum1 = 5;
    int a,b;

    a = sum++;
    b = --sum1;
    printf("%d\n",a);
    printf("%d\n",b);
    printf("%d\n",sum);
    printf("%d\n",sum1);
    printf("%d\n",sum--);
    printf("%d\n",++sum1);
```

```
    return 0;
}
```

6. 输出下列程序的运行结果。

```
#include<stdio.h>
int main(){
    int a,b=322;
    float x,y=8.88f;
    char c1='k',c2;
    a=y;
    x=b;
    a=c1;
    c2=b;
    printf("%d,%f,%d,%c",a,x,a,c2);

    return 0;
}
```

第 3 章　数据输入与输出

在 C 语言中，数据的输入/输出有着相当重要的地位，是程序与用户交互的唯一途径。输入/输出是以计算机为主体而言的，数据输入是用户通过标准输入设备（键盘）将数据传递给程序，数据输出是程序通过标准输出设备（显示器）将数据传递给用户。

C 语言中数据的输入/输出都是由库函数完成的，因此在程序开始时需要使用编译预处理指令“#include <stdio.h>”。

3.1　数据的输入

3.1.1　字符输入函数 getchar( )

字符输入函数 getchar( )的功能是从标准输入设备（键盘）上获得一个字符，一般形式为：“getchar( );”。

通常把输入的字符赋予一个字符变量，构成赋值语句，例如：

```
char ch;
ch= getchar();
```

使用 getchar( )函数时，应注意：

（1）只能接收单个字符，当输入多于一个字符时，只接收第一个字符，多余字符作废；

（2）用户输入回车键后，系统才开始接收字符；

（3）用 getchar( )得到的字符可以赋给字符型变量、整型变量，或作为表达式的一部分，如“putchar (getchar( ));”、“printf("%c", getchar( ));”等。

（4）如果函数正常，将返回接收字符的 ASCII 值；如果出错，则返回 EOF（即–1）。

【例 3-1】 程序 3-1：字符输入函数。

```
#01: //程序 3-1
#02: #include <stdio.h>
#03: int main(){
#04:     char ch;
#05:     printf("Input a upper-case char:");
#06:     ch=getchar()+32;
#07:     printf("%c,hex:%x\n",ch,ch);
#08:
#09:     return 0;
#10: }
```

程序解释：

#06：从标准输入（键盘）获取一个字符，将其 ACSII 值加 32，值赋给变量 ch。由于小写字母比对应的大写字母的 ASCII 值大 32，因此如果输入一个大写字母，则变量 ch 为对应的小写字母的 ASCII 值。

#07：输出变量 ch 的字符形式和十六进制形式。

程序运行结果如下：

```
Input a upper-case char:D
d,hex:64
```

3.1.2　格式输入函数 scanf()

格式输入函数 scanf( ) 按用户指定的格式从标准输入设备（键盘）把数据输入到指定的变量中。

scanf( )函数一般形式为：

scanf("格式控制字符串", 地址列表);

其中“格式控制字符串”用于指定输入格式，不能显示提示字符串；“地址列表”是变量的地址，由地址运算符“&”后跟变量名组成，如“&a, &b”表示变量 a 和变量 b 的地址，需要 scanf( ) 函数输入到几个变量中，地址表列就包含多少个变量的地址。

“&”是一个取地址运算符，&a 是一个表达式，含义是求变量 a 的地址。“地址列表”中的地址是编译器给变量 a、b 在内存中分配的地址，用户不必关心具体的地址是多少，只需关心该地址处存放的值是多少，即变量值。例如，如果有赋值语句“a=123;”，则 a 为变量名，123 为变量的值，&a 为变量 a 的地址（形如 0xFF120000 这样的值）。

scanf( ) 函数本质上是给变量赋值，但要求写变量的地址，即形如&a 的格式，这与赋值语句是不同的。

【例 3-2】 程序 3-2：scanf( )函数。

```
#01: //程序 3-2
#02: #include <stdio.h>
#03: int main(){
#04:     int a,b,c;
#05:
#06:     printf("Input a, b, c:");
#07:     scanf("%d %d %d",&a, &b, &c);
#08:     printf("a=%d,b=%d,c=%d\n",a, b, c);
#09:
#10:     return 0;
#11: }
```

程序解释：

#06：由于 scanf( ) 函数本身不能显示提示串，故先用 printf( ) 语句输出提示用户“Input a,b,c”的字符串。

#07：等待用户输入三个数值，回车结束。在“%d　%d　%d”之间没有用作输入间隔的非格式字符，因此在输入时要用一个以上的空格或回车作为输入数之间的间隔。

程序运行结果如下：

```
Input a, b, c:11 22 33
a=11,b=22,c=33
```

“格式控制字符串”一般形式为“%[输入数据宽度][长度]类型”，其中方括号“[ ]”项为任选项。

1．类型

类型表示输入数据的类型，其字符和意义如表 3-1 所示。

表 3-1 输入数据类型符号和意义

符 号	意 义
d	输入十进制形式的整数
x	输入十六进制形式的整数
o	输入八进制形式的整数
u	输入十进制无符号整数
c	输入单个字符
s	输入字符串
e	以指数形式输入单精度浮点数
f	以小数形式输入单精度浮点数

【例 3-3】 程序 3-3：输入数据类型。

```
#01: //程序 3-3
#02: #include <stdio.h>
#03: int main(){
#04:     int a,b,c;
#05:     char ch;
#06:
#07:     printf("Input a, b, c, ch:");
#08:     scanf("%d %o %x %c",&a, &b, &c,&ch);
#09:     printf("a=%d,b=%d,c=%d,ch=%d,ch=%c\n",a, b, c,ch,ch);
#10:
#11:     return 0;
#12: }
```

程序解释：

#08：输入的数据分别为十进制、八进制、十六进制、字符形式。

#09：按照十进制或字符形式输出。

程序运行结果如下：

```
Input a, b, c, ch:11 11 11 a
a=11,b=9,c=17,ch=97,ch=a
```

2. 宽度

宽度是用十进制整数指定输入数值的宽度（即字符数）。例如，“scanf("%6d",&i);”，当输入“123456789”时，只把“123456”赋予变量 i，其余部分被截去。

【例 3-4】 程序 3-4：输入数据宽度。

```
#01: //程序 3-4
#02: #include <stdio.h>
#03: int main(){
#04:     int year,month,day;
#05:     char ch;
#06:
#07:     printf("Input year month day:");
```

```
#08:    scanf("%4d%2d%2d",&year,&month,&day);
#09:    printf("year=%d,month=%d,day=%d\n",year, month, day);
#10:
#11:    return 0;
#12: }
```

程序解释：

#08：输入数值的前 4 位赋值给变量 year，接着两位赋值给变量 month，再接着的两位赋值给变量 day。

程序运行结果如下：

```
Input year month day:20100803
year=2010,month=8,day=3
```

3．长度

长度格式符为“h”和“l”，“h”表示输入短整型数据，“l”表示输入长整型数据（如“%ld”）或双精度浮点数（如“%lf”），如果是“ll”，则表示长长整型或者长双精度型。

【例 3-5】 程序 3-5：输入数据长度。

```
#01: //程序 3-5
#02: #include <stdio.h>
#03: int main(){
#04:    long long int count;
#05:    double sum;
#06:
#07:    printf("Input long int,double:");
#08:    scanf("%lld %lf",&count,&sum);
#09:    printf("count=%lld,sum=%15.12lf\n",count, sum);
#10:
#11:    return 0;
#12: }
```

程序解释：

#08：分别输入一个长长整型和双精度浮点数。

#09：按长长整型和双精度浮点数输出。

程序运行结果如下：

```
Input long int,double:4294967295 3.14159265358979
count=4294967295,sum= 3.141592653590
```

使用 scanf( )函数应注意以下几点。

（1）scanf( )函数要求给出变量地址，如果指定变量名，则会出错。如“scanf("%d", a);”是非法的，应为“scanf("%d", &a);”。

（2）scanf( )函数不能控制精度。如“scanf("%3.2f",&price);”是非法的，若输入“123”，不可能使“price=1.23”。

（3）输入多个数值时，若没有作输入数据间隔的非格式字符，则可以用空格、Tab 键或回车键作为数据之间的间隔。

（4）如果“格式控制字符串”中有非格式字符（如“,”、“#”等），则输入时也要输入对应的非格式字符。如“scanf("%d, %d",&a, &b);”，其中用字符“,”作间隔符，则输入时应为“1,2”的形式。

【例 3-6】 程序 3-6：输入数据中含非格式字符。

```
#01: //程序 3-6
#02: #include <stdio.h>
#03: int main(){
#04:    int year,month,day;
#05:    int hour,minute,second;
#06:
#07:    printf("Input year-month-day:");
#08:    scanf("%d-%d-%d",&year,&month,&day);
#09:    printf("year=%d,month=%d,day=%d\n",year, month, day);
#10:
#11:    fflush(stdin);
#12:    printf("Input hour:minute:second:");
#13:    scanf("%d:%d:%d",&hour, &minute, &second);
#14:    printf("hour=%d,minute=%d,second=%d\n",hour, minute, second);
#15:    return 0;
#16: }
```

程序解释：

#08：输入数据之间要求有“–”。

#11：清除输入缓冲区，将第一次输入的回车清除掉，以免影响下次输入。

#13：输入数据之间要求有“:”。

程序运行结果如下：

```
Input year-month-day:2010-8-3
year=2010,month=8,day=3
Input hour:minute:second:12:30:36
hour=12,minute=30,second=36
```

（5）如果输入的数据存在非法数据，则输入结束。如“scanf("%d",&start);”，若输入“123x”，由于“x”为非数值，结束输入，从而“start=123”。

（6）输入字符数据时，若“格式控制字符串”中无非格式字符，则所有输入的字符均为有效字符，空格和转义字符也作为有效字符输入。如“scanf("%c%c",&ch1, &ch2);”，如果输入“a b”，则把字符'a'赋予变量 ch1，把字符'b'赋予变量 ch2。正确的输入方法为“ab”。

【例 3-7】 程序 3-7：输入字符数据。

```
#01: //程序 3-7
#02: #include <stdio.h>
#03: int main(){
#04:    char ch1,ch2;
#05:
#06:    printf("input two char:");
#07:    scanf("%c%c",&ch1,&ch2);
#08:    printf("ch1=%c,ch2=%c(%d)\n",ch1, ch2,ch2);
```

```
#09:
#10:    fflush(stdin);
#11:    printf("input two char:");
#12:    scanf("%c %c",&ch1,&ch2);
#13:    printf("ch1=%c,ch2=%c\n",ch1, ch2);
#14:
#15:    return 0;
#16: }
```

程序解释：

#07：输入的字符'x'赋给变量 ch1，接下来的空格（ASCII 值为 32）赋给变量 ch2。

#12：由于两个“%c”之间存在空格，因此输入的字符“x y”中第一个字符 'x' 赋给变量 ch1，第二个字符 'y' 赋给变量 ch2。

程序运行结果如下：

```
input two char:x y
ch1=x,ch2= (32)
input two char:x y
ch1=x,ch2=y
```

（7）如果输入数据与输出类型不一致，可能会导致结果错误。

【例 3-8】 程序 3-8：输入数据与输出数据类型不一致。

```
#01: //程序 3-8
#02: #include <stdio.h>
#03: int main(){
#04:    int a;
#05:    long long b;
#06:
#07:    printf("Input a int:");
#08:    scanf("%d",&a);
#09:    printf("%lld\n",a);
#10:    printf("Input a long long int:");
#11:    scanf("%lld",&b);
#12:    printf("%lld\n",b);
#13:
#14:    return 0;
#15: }
```

程序解释：

#08, #09：输入数据类型为整型，而输出语句格式串中说明为长长整型，因此输出结果和输入数据不符。

#11, #12：输入数据类型为长长整型，输出语句格式串中说明为长长整型，因此输出结果和输入数据相同。

程序运行结果如下：

```
Input a int:4294967295
115395506271682555
Input a long long int:4294967295
4294967295
```

（8）scanf( ) 读取变量值时，如果遇到回车结束输入，则回车字符（ASCII 值为 10）会留在输入缓冲区中，形成“垃圾”。如果再次使用数据输入函数读取字符时，会将其当作字符读入。因此在 scanf( ) 调用后需要将回车字符清除掉，推荐使用“fflush(stdin);”的方法。

fflush( ) 函数在头文件“stdio.h”中定义，作用是清空输入缓冲区，通常为了确保不影响后面的数据读取，如在读完一个字符串后紧接着又要读取一个字符，此时应该先执行“fflush(stdin);”。

【例 3-9】 程序 3-9：清除回车字符。

```
#01: //程序 3-9
#02: #include <stdio.h>
#03: int main(){
#04:     int x;
#05:     char ch;
#06:
#07:     scanf("%d",&x);
#08:     ch=getchar();
#09:     printf("x=%d,ch=%d\n",x,ch);
#10:
#11:     scanf("%d",&x);
#12:     fflush(stdin);
#13:     ch=getchar();
#14:     printf("x=%d,ch=%d\n",x,ch);
#15:
#16:     return 0;
#17: }
```

程序解释：

#07, #08：输入数值 123 后，回车作为字符被 getchar( )函数读取到 ch 中。

#09：输出 ch 的值为 10，即回车的 ASCII 值。

#12：清空当前输入缓冲区中的内容，即回车。

#13：输入缓冲区中无内容，因此等待用户输入一个字符。

#14：正确显示了用户输入的数值和字符。

程序运行结果如下：

```
123
x=123,ch=10
123
a
x=123,ch=97
```

3.2 数据的输出

3.2.1 字符输出函数 putchar( )

字符输出函数 putchar( )功能是在标准输出设备（显示器）上输出单个字符，一般形式为“putchar(ch)”，其中 ch 为字符常量、字符变量或整型变量。如果函数正常，将输出 ch 的值；如果出错，则返回 EOF（即–1）。

【例 3-10】 程序 3-10：字符输出函数。

```
#01: //程序 3-10
#02: #include <stdio.h>
#03: int main(){
#04:     char ch='A';
#05:     int x=65;
#06:
#07:     putchar('A');
#08:     putchar(x);
#09:     putchar(ch);
#10:     putchar('\101');
#11:     putchar('\\');
#12:
#13:     return 0;
#14: }
```

程序解释：

#07：输出字符'A'。

#08：输出整型变量 x 的值对应的字符。

#09：输出字符变量 ch 的值对应的字符。

#10：输出字符常量，101 为八进制数，对应于十进制数 65，即'A'的 ASCII 值。

#11：输出字符常量，表示转义字符“\”。

程序运行结果如下：

```
AAAA\
```

3.2.2　格式输出函数 printf( )

格式输出函数 printf( )将指定格式的数据显示到标准输出设备（显示器）上，标识符 printf 最末一个字母 f 即为“格式”（format）之意，即按用户指定的格式。

printf( )函数一般形式为“printf(“格式控制字符串”, 输出表列);”，其中“格式控制字符串”用于指定输出格式，“输出表列”给出了各个要输出的数据（常量、变量或表达式），“输出表列”可以没有，也可同时输出多个并以逗号“,”分隔。

“格式控制字符串”是用双引号括起来的字符串，由“格式说明字符串”和“非格式说明字符串”两类组成。“格式说明字符串”以%开头，在%后面跟有各种格式字符，以说明输出数据的类型、形式、长度、小数位数等。“非格式说明字符串”为普通字符或转义序列，将字符原样输出，在显示中起提示作用。

“格式说明字符串”用于指定输出格式，如“%d”表示按十进制整数输出、“%c”表示按字符型输出等。“格式说明字符串”和“输出表列”在数量和类型上一一对应。

【例 3-11】 程序 3-11：格式输出函数。

```
#01: //程序 3-11
#02: #include <stdio.h>
#03: int main(){
#04:     int a=88,b=89;
#05:
#06:     printf("%d %d\n",a, b);
```

```
#07:    printf("%d,%d\n",a, b);
#08:    printf("%c %c\n",a, b);
#09:    printf("a=%d, b=%d\n",a, b);
#10:
#11:    return 0;
#12: }
```

程序解释：

#06：“格式控制字符串”为“%d”，则将变量 a 和 b 用整数形式输出。两格式串“%d”之间有一个空格（非格式说明字符串），所以输出的 a、b 值之间有一个空格。

#07：两格式串“%d”之间有一个逗号（非格式说明字符串），所以输出的 a、b 值之间有一个逗号。

#08：“格式控制字符串”为“%c”，则将变量 a 和 b 用字符形式输出。

#09：“格式控制字符串”中的“a=”和“,b=”为“非格式说明字符串”，则原样输出。

程序运行结果如下：

```
88 89
88,89
X Y
a=88, b=89
```

“格式说明字符串”的一般形式为“%[修饰符]格式类型”，其中“修饰符”包括标志、输出最小宽度、精度、长度等，因此一般形式可以表示为“%[标志][输出最小宽度][.精度][长度]类型”，其中方括号“[ ]”中的项为可选项。

1．格式类型

格式类型用以表示输出数据的类型，其符号意义如表 3-2 所示。

表 3-2　格式类型的符号意义

符　号	意　义	示　例	输 出 结 果
d 或 i	以十进制形式输出带符号整数（正数不输出符号）	int a=987; printf(“%d”, a);	987
x 或 X	以十六进制形式输出无符号整数（不输出前缀 0x）	int a=–1; printf(“%x”, a);	ffffffff
o	以八进制形式输出无符号整数（不输出前缀 0）	int a=65; printf(“%o”, a);	101
u	以十进制形式输出无符号整数	int a=–1; printf(“%u”, a);	4294967295
c	输出单个字符	char a=65; printf(“%c”, a);	A
s	输出字符串	printf(“%s”, “ABC”);	ABC
e 或 E	以指数形式输出单、双精度实数	double a=567.789; printf(“%e”, a);	5.677890e+002
f	以小数形式输出单、双精度实数	double a=567.789; printf(“%f”, a);	567.789000
g 或 G	以%f 或%e 中较短的输出宽度输出单、双精度实数	double a=567.789; printf(“%g”, a);	567.789
%%	输出百分号本身	printf(“%%”);	%

【例 3-12】 程序 3-12：格式输出函数格式类型。

```
#01: //程序 3-12
#02: #include <stdio.h>
#03: int main(){
#04:     int i=99;
#05:     double d=3.14;
#06:     int a=-1;
#07:
#08:     printf("i=%d,i=%i,i=%o,i=%x,i=%c\n", i,i,i,i,i);
#09:     printf("d=%e,d=%f,d=%g\n",d,d,d);
#10:     printf("a=%d,a=%o,a=%x,a=%u\n",a,a,a,a);
#11:
#12:     return 0;
#13: }
```

程序解释：

#08：整型变量 i 分别以十进制、十进制、八进制、十六进制、字符输出。

#09：实型变量 d 分别以指数形式、小数形式、两者较短形式输出。

#10：整型变量 d 分别以十进制、八进制、十六进制、无符号整数输出。

程序运行结果如下：

```
i=99,i=99,i=143,i=63,i=c
d=3.140000e+000,d=3.140000,d=3.14
a=-1,a=37777777777,a=ffffffff,a=4294967295
```

2．输出最小宽度

用一个十进制整数来指定输出数据的最少位数，若实际位数多于指定宽度，则按实际位数输出，若实际位数少于指定宽度，则补以空格或 0（由是否有标志“0”决定）。

3．精度

精度以“.”开头，后跟一个十进制整数，其意义是：如果输出数值，则表示数值小数部分的位数（四舍五入）；如果输出字符串，则表示输出字符的个数；若实际位数大于指定精度，则截去超出的部分。

4．标志

标志符号有“–”、“+”、“ ”（空格）、“0”、“#”这 5 种，其意义如表 3-3 所示。

表 3-3　标志符号意义

符　号	功　能
–	输出数据左对齐，右边填空格（默认为右对齐）
+	输出正负号，在有符号数的正数前显示正号（+）
（空格）	输出值为正数时显示空格，为负数时显示负号
0	输出数值右对齐时，左面的空位置自动填 0，默认为空格
#	在八进制数和十六进制数前分别显示前导“0”和“0x”；对于实型数，当结果有小数时，才给出小数点

【例 3-13】 程序 3-13：格式输出函数标志。

```
#01: //程序 3-13
#02: #include <stdio.h>
#03: int main(){
#04:     int a=123;
#05:     float f=123.456;
#06:
#07:     printf("a=%8d\n",a);
#08:     printf("a=%08d\n",a);
#09:     printf("a=%0+8d\n",a);
#10:     printf("f=%10.2f\n",f);
#11:     printf("f=%010.2f\n",f);
#12:     printf("f=%0+10.2f\n",f);
#13:     printf("a=%o,a=%#o,a=%x,a=%#x\n",a,a,a,a);
#14:
#15:     return 0;
#16: }
```

程序解释：

#07：变量 a 输出宽度为 8 位，右对齐。

#08：变量 a 输出宽度为 8 位，右对齐，空位填 0。

#09：变量 a 输出宽度为 8 位，右对齐，空位填 0，正数加正号+。

#10：变量 f 输出宽度为 10 位（含小数点），其中小数部分占两位，右对齐。

#11：变量 f 输出宽度为 10 位（含小数点），其中小数部分占两位，右对齐，空位填 0。

#12：变量 f 输出宽度为 10 位（含小数点、正负号），其中小数部分占两位，右对齐，空位填 0，正数加正号+。

#13：变量 a 分别输出八进制数、带前缀 0 的八进制数、十六进制数、带前缀 0x 的十六进制数。

程序运行结果如下：

```
a=     123
a=00000123
a=+0000123
f=    123.46
f=0000123.46
f=+000123.46
a=173,a=0173,a=7b,a=0x7b
```

5．长度

长度符号有“h”、“l”两种，其意义如表 3-4 所示。

其中，“h”表示按短整型量输出，l 表示按长整型量输出，如果是“ll”，则表示长长整型或长双精度型。

表 3-4 长度符号意义

符 号	功 能
h	按短整型量输出
l	对于整型数，指定输出精度为 long 型
	对于实型数，指定输出精度为 double 型

【例 3-14】 程序 3-14：格式输出函数长度。

```
#01: //程序 3-14
#02: #include <stdio.h>
#03: int main(){
#04:    long long a=4294967296;
#05:    float f=123.1234567;
#06:    double d=123456789.123456789;
#07:
#08:    printf("a=%lld,a=%d\n", a,a);
#09:    printf("f=%f,f=%lf,f=%5.4lf,f=%e\n",f, f, f, f);
#10:    printf("d=%lf,d=%f,d=%8.1lf\n",d, d, d);
#11:
#12:    return 0;
#13: }
```

程序解释：

#08：输出为长长整型时输出正常。输出为整型时，截取高 4 字节，只剩下低 4 字节（全零），从而输出零。

#09：格式“%f”和“%lf”输出相同，说明“l”对单精度浮点数无影响。“%5.4lf”指定输出宽度为 5，精度（小数部分）为 4，由于实际长度超过 5，故按实际位数输出，小数位数超过 4 的部分被截去。

#10：格式“%8.1lf”指定精度为 1，截去了小数部分超过 1 位的部分。

程序运行结果如下：

```
a=4294967296,a=0
f=123.123459,f=123.123459,f=123.1235,f=1.231235e+002
d=123456789.123457,d=123456789.123457,d=123456789.1
```

3.3　顺序结构程序设计

从程序流程的角度可以将程序分为三种基本结构：顺序结构、选择结构、循环结构，各种复杂的程序都可以由这三种基本结构组成。在 1.3.3 节中已对这三种基本结构做了描述。

顺序结构的程序设计是最简单的，只要按照解决问题的顺序写出相应的语句即可，它的执行顺序是自上而下，依次执行每条语句。顺序结构可以独立构成一个简单的完整程序，常见的输入、计算、输出三部曲的程序就是顺序结构，如计算圆的面积，程序的语句顺序是：①输入圆的半径 r；②计算 area = 3.14159∗r∗r；③输出圆的面积 area。

大多数情况下顺序结构都是作为程序的一部分，与其他结构一起构成一个复杂的程序，如选择结构中的复合语句、循环结构中的循环体等。

【例 3-15】 程序 3-15：顺序结构程序设计，输入长方形的两个边长，计算长方形的周长和面积。

我们分析该程序的流程，程序只要被告知长方形的长和宽，就能够正确得出长方形的周长和面积。因此程序第一个步骤是输入长方形的长和宽。

得到了长和宽，根据数学计算公式：周长=2×(长+宽)、面积=长×宽，程序可以计算出周长和面积，因此程序第二个步骤是计算。

程序应该将计算结果通知用户，因此程序第三个步骤是输出周长和面积。

从而，我们要实现的程序的流程是：输入→计算→输出，流程图如图 3-1 所示。

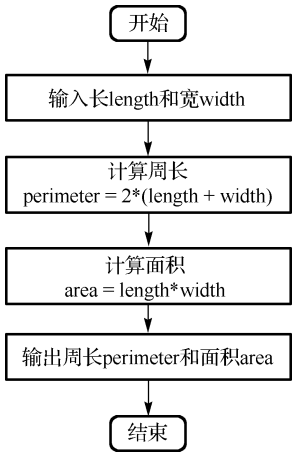


图 3-1　计算长方形周长和面积的流程图

程序如下：

```
#01: //程序 3-15
#02: #include <stdio.h>
#03: int main(){
#04:     double length,width;
#05:     double perimeter,area;
#06:
#07:     printf("Input rectangle length:");
#08:     scanf("%lf", &length);
#09:     printf("Input rectangle width:");
#10:     scanf("%lf", &width);
#11:
#12:     perimeter = (length+width)*2;
#13:     area = length*width;
#14:
#15:     printf("Rectangle perimeter:%lf\n", perimeter);
#16:     printf("Rectangle area:%lf\n", area);
#17:
#18:     return 0;
#19: }
```

程序解释：

#08, #10：输入长方形的长和宽。

#12, #13：计算长方形的周长和面积。

#15, #16：输出周长和面积。

程序运行结果如下：

```
Input rectangle length:5.2
Input rectangle width:2.5
Rectangle perimeter:15.400000
Rectangle area:13.000000
```

3.4　程 序 示 例

【例 3-16】程序 3-16：求 $ax^2+bx+c=0$ 方程的根，a、b、c 由键盘输入，输入时应满足 $b^2-4ac>0$。由输入的 a、b、c 值，可求出 $p=-\dfrac{b}{2a}$，$q=\dfrac{\sqrt{b^2-4ac}}{2a}$，则 $x_1=p+q$，$x_2=p-q$。

```
#01: //程序 3-16
#02: #include <stdio.h>
#03: #include <math.h>
#04: int main(){
#05:     double a,b,c;
#06:     double p,q,x1,x2;
#07:
#08:     printf("Input a,b,c:");
#09:     scanf("%lf %lf %lf",&a,&b,&c);
#10:     p=-b/(2*a);
#11:     q=sqrt(b*b-4*a*c)/(2*a);
#12:     x1=p+q;
#13:     x2=p-q;
#14:     printf("x1=%8.5lf,x2=%8.5lf\n",x1,x2);
#15:
#16:     return 0;
#17: }
```

程序解释：

#03：因为在#11 行中要用到求平方根函数 sqrt( )，因此要包含头文件 math.h。

#11：利用函数 sqrt( )求平方根。

程序运行结果如下：

```
Input a,b,c:2 5 3
x1=-1.00000,x2=-1.50000
```

【例 3-17】 程序 3-17：摄氏温度、华氏温度转换程序。

转换公式：fahrenheit =(celsius +32)*9/5。

```
//程序 3-17
#include <stdio.h>
int main(){
   double celsius,fahrenheit;

   printf("Input celsius:");
   scanf("%lf",&celsius);
   fahrenheit=(celsius+32.0)*9.0/5.0;
   printf("fahrenheit=%.2lf\n",fahrenheit);

   return 0;
}
```

程序运行结果如下：

```
Input celsius:37.5
fahrenheit=125.10
```

上机实验：顺序结构程序设计应用

本次实验掌握 C 语言程序的顺序结构，熟练应用赋值、输入、输出语句。

（1）输入小写字母，转换为大写字母输出。

```
#include <stdio.h>
int main(){
   char ch;

   printf("Input a low-case char:");
   ch=getchar();
   printf("%c,%d\n",ch,ch);
   ch=ch-32;
   printf("%c,%d\n",ch,ch);

   return 0;
}
```

（2）输入三角形的三边长，求三角形的面积。

三角形面积公式：$\text{area}=\sqrt{p(p-a)(p-b)(p-c)}$，其中 $p=\frac{1}{2}(a+b+c)$。

```
#include <stdio.h>
#include <math.h>
int main(){
   double a,b,c,p,area;

   printf("Input triangle a b c:");
   scanf("%lf %lf %lf",&a,&b,&c);
   p=1.0/2*(a+b+c);
   area=sqrt(p*(p-a)*(p-b)*(p-c));
   printf("a=%5.2f,b=%5.2f,c=%5.2f\n",a,b,c);
   printf("area=%5.2f\n",area);

   return 0;
}
```

习　　题

1．编写程序，从键盘上输入两个电阻的值，求它们并联和串联的电阻值，输出结果保留两位小数。

2．编写程序，从键盘输入梯形的上、下底边的长度和高，计算梯形的面积。

3．编写程序，从键盘输入一个字符，求出与该字符前后相邻的两个字符，按从小到大的顺序输出这三个字符的 ASCII 码。

4．编写程序，从键盘输入圆半径、圆柱高，计算圆周长、圆柱体积。

5．编写程序，从键盘输入三角形的底边及高的长度，求其面积。

6．编写程序，从键盘输入圆的半径值，求圆的周长和面积。

7．编写程序，从键盘输入球体的半径，求球体的体积和表面积。

第 4 章　控制结构程序设计

4.1　关系运算符与逻辑运算符

关系运算是将两个值进行比较，判断比较结果是否符合条件。比较两个量大小关系的运算符称为关系运算符，如大于关系运算符为“>”。

逻辑运算符把多个关系连接成更复杂的关系，如水的温度 temp 在 0 和 100 之间，用逻辑运算符表示为：“(temp>0) &&(100>temp)”，其中“&&”表示逻辑运算符“与”，表示“temp>0”和“100>temp”同时成立。

4.1.1　关系运算符

关系运算符有 6 种，如表 4-1 所示。

表 4-1　关系运算符

关系运算符	意　义	示　例
<	小于	a>(b+c)
<=	小于等于	a<=(b+c)
>	大于	a>(b+c)
>=	大于等于	a>=(b+c)
= =	等于	a= =(b+c)
!=	不等于	a!=(b+c)

关系运算符为双目运算符，需要两个操作数，结合性为左结合。

关系运算符的优先级低于算术运算符，高于赋值运算符。在 6 个关系运算符中，“<”、“<=”、“>”和“>=”的优先级相同，高于“= =”和“!=”（此两种运算符优先级相同）。例如，“c>a+b”与“c>(a+b)”等价，“a>b!=c”与“(a>b)!=c”等价，推荐在关系运算中添加括号“( )”来表示不同的关系运算，如“a=(b>c)”。

关系表达式是用关系运算符将两个数值或者数值表达式连接起来的式子，一般形式为：“表达式 关系运算符 表达式”，如“(a+b)>(c–d)”、“x>3/2”、“('a'+1)<c”、“i–(5*j)==(k+1)”等。

关系表达式一般形式中的表达式也可以是关系表达式，因此会出现嵌套的情况，如“a>(b>c)”、“a!=(c= =d)”等。

关系表达式的值是“真”或“假”，用“1”和“0”表示。如“5>0”成立，则值为“真”，即 1，“(a=3)>(b=5)”不成立，故值为“假”，即 0。

使用关系运算符应注意如下事项。

（1）不能将实型数用“= =”或“!=”与任何数字比较。无论是 float 还是 double 类型的变量，都有精度限制，所以要避免将浮点变量用“==”或“!=”与数字比较，应该设法转化成“>=”或“<=”形式。

例如，float 变量 x 与“零值”比较应写为：(x >= –1e–6) && (x <= 1e–6)，其中 1e–6 是允许的误差（精度）。

（2）“=”与“==”的意义是不同。例如，定义“int a=0,b=1;”，“a=b”表示将变量 b 的值赋值给变量 a，结果为 1。“a==b”表示比较变量 a 和变量 b 的值是否相同，结果为 0。

【例 4-1】 程序 4-1：关系运算符。

```
#01: //程序 4-1
#02: #include <stdio.h>
#03: int main(){
#04:    float a=0, b=0.5, x=0.3;
#05:    int  i=1, j=7,k;
#06:
#07:    printf("a<=x<=b:%d\n",a<=x<=b);
#08:    printf("5>2>7>8:%d\n",5>2>7>8);
#09:    printf("' i+(j%4!=0):%d\n", i+(j%4!=0));
#10:    printf("'a'>0:%d\n",'a'>0);
#11:    printf("'A'>100:%d\n",'A'>100);
#12:
#13:    return 0;
#14: }
```

程序解释：

#07：“a<=x<=b”等价于“(a<=x)<=b”，先判断“a<=x”是否成立（成立为 1），再判断“1<=b”是否成立（不成立为 0）。

#08：“5>2>7>8”等价于“((5>2)>7)>8”，判断后不成立，为 0。

#09：“j%4!=0”等价于“(j%4)!=0”。

#10, #11：字符变量是以它对应的 ASCII 码参与关系运算的。

程序运行结果如下：

```
a<=x<=b:0
5>2>7>8:0
i+(j!=0):2
'a'>0:1
'A'>100:0
```

4.1.2　逻辑运算符

逻辑运算符对逻辑值（真或假）进行操作，逻辑运算符有三种，如表 4-2 所示。

表 4-2　逻辑运算符

逻辑运算符	意　义	示　例
&&	与运算	a&&b
‖	或运算	a‖b
!	非运算	!a

逻辑运算符中的与运算符“&&”和或运算符“‖”为双目运算符，具有左结合性。非运算符“!”为单目运算符，具有右结合性。

逻辑运算真值表如表 4-3 所示，表示各种逻辑运算的结果。

表 4-3 逻辑运算真值表

a	b	!a	!b	a&&b	a||b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

根据逻辑运算真值表，逻辑运算的求值规则如下。

（1）与运算&&

参与运算的两个量都为真时，结果才为真，否则为假。如“(2>1) && (3>2)”，由于“2>1”为真，“3>2”为真，因此进行与运算后结果也为真。

（2）或运算 ||

参与运算的两个量只要有一个为真，结果就为真；只有当两个量都为假时，结果才为假。如“(2<1) && (3>2)”，由于“2<1”为假，“3>2”为真，因此进行或运算后结果为真。

（3）非运算!

参与运算量为真时，结果为假；参与运算量为假时，结果为真。如“!(2>1)”，由于“2>1”为真，因此进行非运算后结果为假。

逻辑运算的结果为“真”和“假”两种，用“1”和“0”分别表示。结果只能是“1”或“0”，不可能是其他数值。在进行逻辑运算判断一个量是为“真”还是为“假”时，运算量“0”作为“假”，非“0”的数值均作为“真”。如 1 和 2 均为非“0”，因此 1&&2 的值为“真”，即为 1。

逻辑运算符和其他运算符优先级的关系如图 4-1 所示。

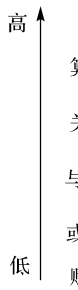


图 4-1 逻辑运算符和其他运算符优先级

从图中可以看出，对于三个逻辑运算符，其优先级顺序为：!（非）→&&（与）→||（或）依次降低。例如，“!a||(a>b)”等价于“(!a)||(a>b)”，“a||b&&c”等价于“a||(b&&c)”。

逻辑运算符“&&”和“||”低于关系运算符，高于赋值运算符；“!”高于算术运算符。例如，“a<=x && x<=b”等价于“(a<=x) && (x<=b) ”，“a==b||x==y”等价于“(a==b)||(x==y)”，“a+b>c&&x+y<b”等价于“((a+b)>c)&&((x+y)<b)”，“!b==c||d<a”等价于“((!b)==c)||(d<a)”。

逻辑表达式是用逻辑运算符将关系表达式或逻辑量连接起来的有意义的式子，一般形式为：“表达式 逻辑运算符 表达式”，如“a&&b”、“!a||b”等。

逻辑表达式一般形式中的表达式也可以是逻辑表达式，因此会出现嵌套的情况，如“a&&b&&c”、“5>3&&2||8<4-!0”等。

逻辑表达式的值是式中各种逻辑运算的最后值，以“1”和“0”分别代表“真”和“假”。

逻辑表达式具有短路特性：对逻辑表达式求解时，并非所有的逻辑运算符都会被执行，只有在必须执行下一个逻辑运算符才能求出表达式的值时，才会执行该运算符。如“a&&b&&c”，只在 a 为真时，才判别 b 的值；只在 a、b 都为真时，才判别 c 的值。如果 a 为假，不对 b 和 c 进行求值；如果 a 为真，b 为假，则不对 c 求值。再如“a||b||c”，只在 a 为假时，才判别 b 的值；只在 a、b 都为假时，才判别 c 的值。如果 a 为真，不对 b 和 c 进行求值；如果 a 为假，b 为真，则不对 c 求值。

【例 4-2】 程序 4-2：逻辑运算符。

```
#01: //程序 4-2
#02: #include <stdio.h>
#03: int main(){
#04:     float x=2e3,y=0.5;
#05:     int a=1,b=2,c=3,d=4;
#06:     int m=1,n=1;
#07:     char ch='a';
#08:
#09:     printf("!x*!y=%d,!!!x=%d\n",!x*!y,!!!x);
#10:     printf("%d, %d\n",x||a&&b-3,a<b&&x<y);
#11:     printf("%d, %d\n",a==5&&ch&&(b=1),x+y||a+b+c);
#12:     (m=a>b)&&(n=c>d);
#13:     printf("m=%d,n=%d\n",m,n);
#14:
#15:     return 0;
#16: }
```

程序解释：

#09：!x 和!y 分别为 0，!x*!y 也为 0。由于 x 非 0，故!!!x 的逻辑值为 0。

#10：先计算 b–3 的值为非 0，再求 a&&b–3 的逻辑值为 1，故表达式的逻辑值为 1。由于 a<b 的值为 1，而 x<y 为 0，故表达式的值为 0。

#11：由于 a==5 为假，即值为 0，该表达式由两个与运算组成，所以整个表达式的值为 0（短路特性）。由于 x+y 的值为非 0，故整个或表达式的值为 1（短路特性）。

#12：先进行 a>b 判断，值为 0，然后赋值给 m。由于是与运算，根据短路特性，“n=c>d”不进行求值，因此 n 保持原值 1 不变。

#13：输出 m=0，n=1，验证短路特性。

程序运行结果如下：

```
!x*!y=0,!!!x=0
1, 0
0, 1
m=0,n=1
```

4.2　选择结构程序

4.2.1　if 语句

用 if 语句可以构成选择结构（或称分支结构），if 语句对给定条件进行判断，以决定执行某个分支程序段。

1．if 语句三种形式

（1）形式一：基本 if 语句

基本 if 语句的格式为：

if (表达式)
　　语句

其含义是：如果表达式的值为真，则执行其后的语句，否则不执行该语句。其过程如图 4-2 所示。

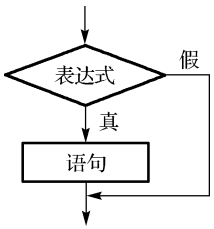


图 4-2 if 语句流程图

例如，下列语句表示：如果变量 x 大于变量 y，则输出变量 x 的值，否则不执行任何语句。

```
if (x>y)
    printf("%d",x);
```

【例 4-3】 程序 4-3：基本 if 语句。

```
#01: //程序 4-3
#02: #include <stdio.h>
#03: int main(){
#04:     int a,b,min;
#05:
#06:     printf("Input two integers:");
#07:     scanf("%d %d",&a,&b);
#08:     min=a;
#09:     if (min>b)
#10:         min=b;
#11:
#12:     printf("Minimum of (%d,%d) is:%d\n",a,b,min);
#13:
#14:     return 0;
#15: }
```

程序解释：

#08：把变量 a 先赋予变量 min。

#09, #10：用 if 语句判别 min 和 b 的大小，如 min 大于 b，则把 b 赋予 min。因此 min 总是 a、b 两者之小数。

程序运行结果如下：

```
Input two integers:3 5
Minimum of (3,5) is:3
```

（2）形式二：if-else 语句

if-else 语句的格式为：

if (表达式)
　　语句 1;
else
　　语句 2;

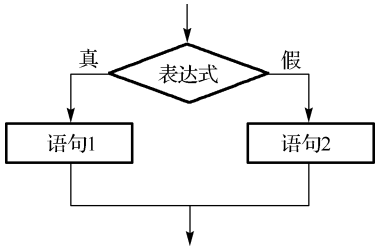


图 4-3 if-else 语句流程图

其含义是：如果表达式的值为真，则执行语句 1，否则执行语句 2。其过程如图 4-3 所示。

例如，下列语句表示：如果变量 x 大于变量 y，则变量 max=x，否则变量 max=y。

```
if (x>y)
    max=x;
```

```
    else
        max=y;
```

【例 4-4】 程序 4-4：if-else 语句。

```
#01: //程序 4-4
#02: #include <stdio.h>
#03: int main(){
#04:     int a,b,difference;
#05:
#06:     printf("Input two integers:");
#07:     scanf("%d %d",&a,&b);
#08:     if (a>b){
#09:         difference=a-b;
#10:         printf("Maximum:%d\n",a);
#11:     }else{
#12:         difference=b-a;
#13:         printf("Maximum:%d\n",b);
#14:     }
#15:     printf("Difference is:%d\n",difference);
#16:
#17:     return 0;
#18: }
```

程序解释：

#08, #09, #10：判别 a 和 b 的大小，若 a 大，则两者之差为大数（a）减去小数（b），并输出 a 作为最大值。

#11, #12, #13：否则两者之差为大数（b）减去小数（a），并输出 b 作为最大值。

程序运行结果如下：

```
Input two integers:12 34
Maximum:34
Difference is:22
```

（3）形式三：if-else-if 语句

当有多个分支需要选择时，可采用 if-else-if 语句，其格式为：

```
if (表达式 1)
    语句 1;
else if (表达式 2)
    语句 2;
…
else if (表达式 n)
    语句 n;
else
    语句 n+1;
```

其含义是：依次判断表达式的值，当出现某个表达式值为真时，则执行其对应的语句。然后跳到

整个 if 语句之外继续执行后续语句。如果所有表达式均为假，则执行语句 n+1，然后继续执行后续语句。其过程如图 4-4 所示。

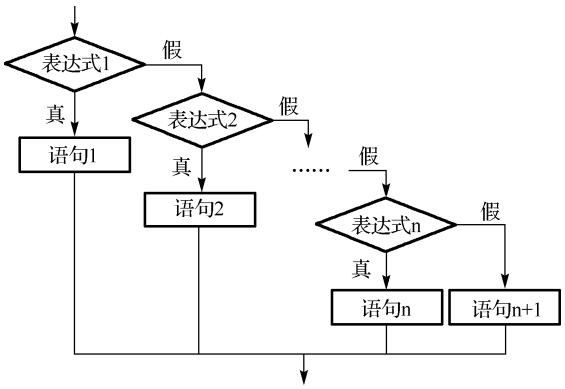


图 4-4 if-else-if 语句流程图

例如，下列语句表示：如果变量 salary 大于 1000，则变量 index=0.4；否则如果变量 salary 大于 800（即 800<salary<=1000），则变量 index=0.3；否则如果变量 salary 大于 600（即 600<salary<=800），则变量 index=0.2；否则如果变量 salary 大于 400（即 400<salary<=600），则变量 index=0.1；否则变量 index=0（此时 salary<=400）。

```
if (salary>1000)
    index=0.4;
else if (salary>800)
    index=0.3;
else if (salary>600)
    index=0.2;
else if (salary>400)
    index=0.1;
else
    index=0;
```

【例 4-5】 程序 4-5：if-else-if 语句，判别键盘输入字符类别。

```
#01: //程序 4-5
#02: #include <stdio.h>
#03: int main(){
#04:     char ch;
#05:
#06:     printf("Input a char:");
#07:     scanf("%c",&ch);
#08:
#09:     if (ch<' ')
#10:        printf("It is a control char,ASCII:%d\n",ch);
#11:     else  if ((ch>='0')&&(ch<='9') )
#12:        printf("It is a digit,ASCII:%d\n",ch);
#13:     else  if ((ch>='A')&&(ch<='Z') )
```

```
#14:        printf("It is a upper-case letter,ASCII:%d\n",ch);
#15:    else  if ((ch>='a')&&(ch<='z') )
#16:        printf("It is a low-case letter,ASCII:%d\n",ch);
#17:    else
#18:        printf("It is other char,ASCII:%d\n",ch);
#19:
#20:    return 0;
#21: }
```

程序解释：

#09, #10：判断输入字符 ASCII 码所在的范围，分别给出不同的输出。由 ASCII 码表可知 ASCII 值小于 32 的字符为控制字符。

#11, #12：ASCII 值在“0”和“9”之间的为数字。

#13, #14：ASCII 值在“A”和“Z”之间的为大写字母。

#15, #16：ASCII 值在“a”和“z”之间的为小写字母。

#17, #18：其余则为其他字符。

程序运行结果如下：

```
Input a char:B
It is a upper-case letter,ASCII:66
```

if 语句在使用中应注意以下事项。

（1）if 语句中 if 关键字之后的条件判断表达式必须用括号括起来，条件判断表达式通常是逻辑表达式或关系表达式，也可以是其他表达式或变量，如赋值表达式。如“if (x=1)”、“if ('a')”、“if (1)”和“if (y)”都是合法的，只要求得表达式的值为非 0，即为“真”，否则为“假”。

（2）if 语句格式中满足条件执行的“语句”可以是一条语句，也可以是多条语句，此时需要用花括号“{}”将其组成一个复合语句，注意在“}”之后不能加分号“;”。例如，下列语句表示：如果变量 x 大于变量 y，则求变量 x 除以 2 的余数并赋值给 x；并且输出变量 x 的值，否则不执行任何语句。

```
if (x>y){
    x=x%2;
    printf("%d", x);
}
```

（3）假设整型变量 x，则“if (x)”等价于“if(x!=0)”，“if(!x)”等价于“if(x==0)”。但是推荐将整型变量用“==”或“!=”直接与 0 比较，否则会让人误解 x 变量的含义。

（4）注意区别类似“if (a=b)”和“if (a==b)”形式的含义。如果存在如下程序：

```
if (a=b)
    printf("%d", a);
else
    printf("a=0");
```

这段程序的意义是：把变量 b 值赋予变量 a，如为非 0，则输出该值，否则输出“a=0”字符串。如果将“if (a=b)”该为“if (a==b)”，则程序意义变为：如果变量 a 与变量 b 相等，则输出 a 值（无论是否为 0），否则输出“a=0”字符串（a 只是与 b 值不同，而不一定为 0）。

2．嵌套 if 语句

当 if 语句中的执行“语句”又是另外一个 if 语句时，就构成了 if 语句嵌套，一般形式为：

```
if (表达式)
    if 语句
```

或

```
if (表达式)
    语句
else
    if 语句;
```

或

```
if (表达式)
    if 语句;
else
    if 语句;
```

在 if 语句中，else 部分不能独立存在，它必定是 if 语句的一部分。

嵌套内的 if 语句有可能又是“if-else”或“if-else-if”形式，这样会出现多个 if 和多个 else 重叠的情况，此时要注意 if 和 else 的配对问题。if～else 配对原则为：没有“{}”时，else 总是和它上面离它最近的未配对的 if 配对。为了能够正确实现 if～else 的配对，可以在适当位置加“{ }”。如下列代码：

```
if（表达式 1）
    if（表达式 2）
    语句 1；
else
    语句 2；
```

应该理解为（添加了花括号“{}”帮助理解）：

```
if（表达式 1）{
    if（表达式 2）
        语句 1；
    else
        语句 2；
}
```

【例 4-6】 程序 4-6：嵌套 if 语句, 比较两个数的大小关系。

```
#01: //程序 4-6
#02: #include <stdio.h>
#03: int main(){
#04:     int x,y;
#05:
#06:     printf("Enter integer x,y:");
#07:     scanf("%d,%d",&x,&y);
```

```
#08:    if(x!=y)
#09:       if(x>y)
#10:          printf("X>Y\n");
#11:       else
#12:          printf("X<Y\n");
#13:    else
#14:      printf("X==Y\n");
#15:
#16:    return 0;
#17: }
```

程序解释：

#09～#12：嵌套 if 语句结构，当 x!=y 时，判断 x>y 是否成立。

#13, #14：此处 else 语句与#08 行的 if 语句相配对。

程序运行结果如下：

```
Enter integer x,y:5,8
X<Y
```

采用嵌套结构是为了进行多分支选择。对于例 4-6，实际上有三种选择，即“A>B”、“A<B”或“A==B”。这类问题用“if-else-if”语句也可以达到目的，而且程序更清晰。因此，一般情况下尽量少用 if 语句的嵌套结构，以使程序更易阅读和理解。

【例 4-7】 程序 4-7：嵌套 if 语句更改为“if-else-if”语句。

```
//程序 4-7
#include <stdio.h>
int main(){
   int x,y;

   printf("Enter integer x,y:");
   scanf("%d,%d",&x,&y);
   if(x==y)
      printf("X==Y\n");
   else if(x>y)
      printf("X>Y\n");
   else
      printf("X<Y\n");

   return 0;
}
```

程序运行结果如下：

```
Enter integer x,y:5,8
X<Y
```

4.2.2　switch 语句

用 if-else-if 语句或嵌套 if 语句可以实现多分支选择，但可读性较差，此时可以使用 switch 语句，一般形式为：

```
switch (表达式){
    case 常量表达式 1: 语句 1;
    case 常量表达式 2: 语句 2;
    …
    case 常量表达式 n: 语句 n;
    default: 语句 n+1;
}
```

其含义是：计算表达式的值，并逐个与常量表达式的值比较，当与某个常量表达式的值相等时，执行其后的语句；然后如果有 break 语句，则跳转出 switch 语句，否则不再判断，继续执行 switch 中该语句之后的所有语句；如表达式的值与所有常量表达式均不相同，则执行 default 后的语句。其过程如图 4-5 所示。

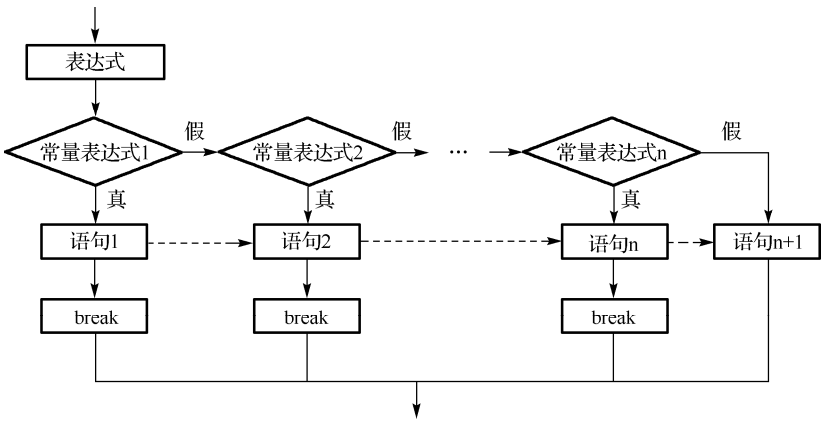


图 4-5 switch 语句流程图

对于 switch 语句应注意以下事项。

（1）表达式可以是整型、字符型或枚举型等；常量表达式的类型必须与表达式的类型相一致（整型与字符型等价）；

（2）常量表达式的值必须唯一，不能相同，先后顺序没有要求，不影响程序的执行结果；

（3）关键字 case 后可包含多条语句，无须加花括号“{}”；

（4）常量表达式仅起语句标号作用，不做求值判断；

（5）default 部分根据需要可以省略不用。

（6）有时可以几种情况（多个 case）公用一组语句，当满足不同常量表达式时，执行相同的语句。如下代码段，当变量 grade 等于'A'、'B' 或 'C'时，都输出“>60”，当变量 grade 等于 'D' 时，输出“<60”。

```
switch (grade){
    case 'A' :
    case 'B':
    case 'C':
        printf(">60\n");
        break;
    case 'D':
```

```
            printf("<60\n");
            break;
    }
```

【例 4-8】 程序 4-8：switch 语句。

```
#01: //程序 4-8
#02: #include <stdio.h>
#03: int main(){
#04:     int x;
#05:
#06:     printf("Enter a integer:");
#07:     scanf("%d",&x);
#08:     switch(x){
#09:        case 1:
#10:           printf("It is Monday.\n");
#11:        case 2:
#12:           printf("It is Tuesday.\n");
#13:        case 3:
#14:           printf("It is Wednesday.\n");
#15:        case 4:
#16:           printf("It is Thursday.\n");
#17:        case 5:
#18:           printf("It is Friday.\n");
#19:        case 6:
#20:           printf("It is Saturday.\n");
#21:        case 7:
#22:           printf("It is Sunday.\n");
#23:        default:
#24:           printf("Input error!\n");
#25:     }
#26:
#27:     return 0;
#28: }
```

程序解释：

#15, #16：程序根据输入的整数，输出对应的星期字符串。当输入“4”时，程序从该行开始执行。

#17～#24：“case 常量表达式”只相当于一个语句标号，若表达式的值和某标号相等，则转向该标号执行，但不能在执行完该标号的语句后自动跳出整个 switch 语句，所以会继续执行后面的所有语句。

程序运行结果如下：

```
Enter a integer:4
It is Thursday.
It is Friday.
It is Saturday.
It is Sunday.
Input error!
```

【例 4-9】 程序 4-9：改进的 switch 语句。

```
#01: //程序 4-9
#02: #include <stdio.h>
```

```
#03: int main(){
#04:    int x;
#05:
#06:    printf("Enter a integer:");
#07:    scanf("%d",&x);
#08:    switch(x){
#09:       case 1:
#10:          printf("It is Monday.\n");
#11:          break;
#12:       case 2:
#13:          printf("It is Tuesday.\n");
#14:          break;
#15:       case 3:
#16:          printf("It is Wednesday.\n");
#17:          break;
#18:       case 4:
#19:          printf("It is Thursday.\n");
#20:          break;
#21:       case 5:
#22:          printf("It is Friday.\n");
#23:          break;
#24:       case 6:
#25:          printf("It is Saturday.\n");
#26:          break;
#27:       case 7:
#28:          printf("It is Sunday.\n");
#29:          break;
#30:       default:
#31:          printf("Input error!\n");
#32:    }
#33:
#34:    return 0;
#35: }
```

程序解释：

#11：break 语句用于跳出 switch 语句，break 语句只有关键字 break，没有参数。

#18：当输入“4”时，程序从该行开始执行。

#20：执行后跳出 switch 语句。

程序运行结果如下：

```
Enter a integer:4
It is Thursday.
```

4.2.3 条件运算符

在选择结构中，当满足或不满足条件都只需执行单个的赋值语句时，可使用条件运算符来实现，不但程序简洁，而且可提高运行效率。

条件运算符为“?”和“:”的组合，是一个三目运算符，有三个运算量，由条件运算符组成条件表达式的一般形式为：

“表达式 1?表达式 2:表达式 3”

其含义是：如果表达式 1 的值为真，则执行表达式 2，并以表达式 2 的值作为条件表达式的值，否则执行表达式 3，并以表达式 3 的值作为整个条件表达式的值。其过程如图 4-6 所示。

条件表达式常用在赋值语句中，例如，语句“min=(a<b)?a:b;”表示：如 a<b 成立，则把变量 a 赋给变量 min，否则把变量 b 赋给变量 min。与下列 if 语句等价。

```
if(a<b)
    min=a;
else
    min=b;
```

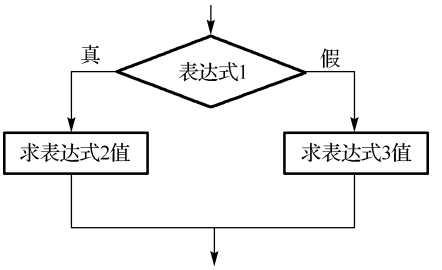


图 4-6　条件表达式流程图

条件表达式在使用时应注意以下事项。

（1）条件运算符“?”和“:”是一对运算符，不能分开单独使用。

（2）条件运算符的优先级低于关系运算符和算术运算符，高于赋值运算符。如“min=(a<b)?a:b;”等价于“min=a<b?a:b;”。

（3）条件运算符可嵌套。如“x>0?1:(x<0?–1:0)”，其中的表达式 3 又是一个条件表达式。

（4）条件运算符的结合方向是自右至左。如“a>b?a:c>d?c:d”等价于“a>b?a:(c>d?c:d)”。

（5）表达式 1、表达式 2、表达式 3 的类型可以不同，此时表达式值取较高的类型。如“x>y?1:1.5”表示当“x>y”时，值为 1.0，否则值为 1.5。

（6）条件表达式不能取代一般的 if 语句，只有当 if 的两个分支为给同一变量赋值时才可取代 if。如下列语句可用条件表达式“printf("%d", a>b?a:b);”表示。

```
if (a>b)
    printf("%d", a);
else
    printf("%d", b);
```

【例 4-10】 程序 4-10：条件运算符。

```
#01: //程序 4-10
#02: #include <stdio.h>
#03: int main(){
#04:     char ch;
#05:
#06:     printf("Enter a char:");
#07:     scanf("%c",&ch);
#08:     ch=(ch>='a'&&ch<='z')?(ch-'a'+'A'):ch;
#09:     printf("ch=%c\n", ch);
#10:
#11:     return 0;
#12: }
```

程序解释：

#08：如果 ch 为小写字母，则转换为大写字母，否则保持 ch 不变。

程序运行结果如下：

```
Enter a char:h
ch=H
```

4.2.4 选择结构程序设计

选择结构是程序设计的三种基本结构之一。顺序结构程序虽然能解决输入、计算和输出等问题，但不能做判断再选择。对于要先做判断再选择的问题，就要使用选择结构，选择结构用于判断给定的条件是否成立，根据判断结果来控制程序流程。

选择结构的执行是依据一定的条件选择执行路径的，而不是严格按照语句出现的先后顺序。选择结构程序设计的关键在于构造合适的选择条件和程序流程，根据不同的程序流程选择适当的选择语句。选择结构适合于带有逻辑或关系比较等条件判断的计算，设计这类程序时往往都要先绘制其程序流程图，然后根据程序流程图写出源程序，这样使得问题简单化，易于理解。

【例 4-11】 程序 4-11：输入三个整数，输出最大数和最小数。

分析该程序的流程，程序中需要依次对三个整数比较大小，根据比较结果选择不同的整数作为最大数和最小数。

首先比较第一个数 x 和第二个数 y，根据大小关系可以找到两个数中的最大数和最小数。

然后将最大数与第三个数 z 比较，如果 z 比此时的最大数还大，则 z 为最大数。否则将最小数与 z 比较，如果 z 比此时的最小数还小，则 z 为最小数。

从而，我们要实现的程序的流程是：先比较大小，然后选择合适的数作为最大数和最小数，流程图如图 4-7 所示。

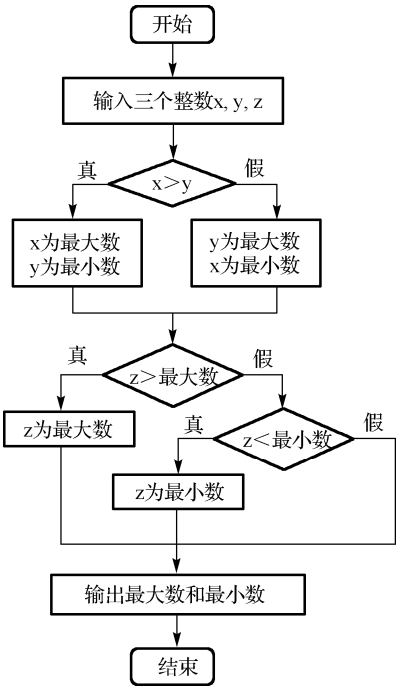


图 4-7 输出三个整数中最大数和最小数流程图

程序如下：

```
#01: //程序 4-11
```

```
#02: #include <stdio.h>
#03: int main(){
#04:    int x,y,z;
#05:    int max,min;
#06:
#07:    printf("Enter 3 integers:");
#08:    scanf("%d %d %d",&x,&y,&z);
#09:
#10:    if (x>y){
#11:       max=x;
#12:       min=y;
#13:    }else{
#14:       max=y;
#15:       min=x;
#16:    }
#17:
#18:    if (z>max)
#19:       max=z;
#20:    else if (z<min)
#21:       min=z;
#22:
#23:    printf("Max=%d,Min=%d\n",max,min);
#24:
#25:    return 0;
#26: }
```

程序解释：

#10～#15：比较输入变量 x 和变量 y 的大小，把大数赋给 max，小数赋给 min。

#18～#19：如果 z 大于 max，则把 z 赋给 max，因此 max 内总是最大数。

#20～#21：如果 z 小于 min，则把 z 赋给 min，因此 min 内总是最小数。

程序运行结果如下：

```
Enter 3 integers:1 2 3
Max=3,Min=1
```

4.3　循环结构程序

循环的本质是不断地重复某种动作。循环结构的特点是：在给定条件成立时，反复执行某个程序段，直到条件不成立为止。给定条件称为“循环条件”，反复执行的程序段称为“循环体”。C 语言提供了三种循环语句，可以组成不同形式的循环结构。

循环结构必须具备两个要求：①在“循环条件”下，重复执行“循环体”；②必须出现不满足条件的情况，使循环终止。

4.3.1　while 与 do-while 语句

while 语句是先判断“循环条件”后执行“循环体”，do-while 语句是先执行“循环体”后判断“循环条件”。

1．while 语句

while 语句的格式为：

while (表达式)
　　循环体语句

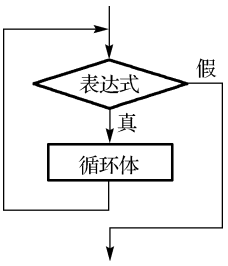


图 4-8　while 语句流程图

其含义是：如果表达式（循环条件）的值为真，则执行循环体语句，然后继续判断表达式是否为真，如果表达式为真，则继续循环；如果表达式为假，则终止循环。其过程如图 4-8 所示。

例如，下列语句表示：只要从键盘输入的字符不是回车键，就继续循环，在循环体内对输入字符的个数进行计数，从而实现对输入字符进行计数的功能。

```
int num=0;
printf("input a string:\n");

while (getchar()!='\n')
   num++;

printf("%d",num);
```

对于 while 语句应注意：

（1）while 语句是先判断表达式，后执行循环体，因此循环体有可能一次也不执行；

（2）while 语句中的表达式一般是关系表达式或逻辑表达式，只要表达式的值为真（非 0），即可继续循环；

（3）循环体可以为任意类型的语句，如果有多条语句，用花括号“{}”组成复合语句；

（4）退出 while 循环的条件有两种情况：①表达式不成立（为假）；②在循环体内遇到 break、return 等语句；

（5）“while(1)”可以表示无限循环。

【例 4-12】 程序 4-12：while 语句，求 1+2+3+…+100 的值。

```
#01: //程序 4-12
#02: #include <stdio.h>
#03: int main(){
#04:     int i,sum;
#05:
#06:     i=1;
#07:     sum=0;
#08:     while (i<101){
#09:         sum+=i;
#10:         i++;
#11:     }
#12:
#13:     printf("1+2+3+...+100=%d\n",sum);
#14:     return 0;
#15: }
```

程序解释：

#08：i 作为循环变量，当小于 101 时执行循环体。

#10：修改循环变量 i 的值，以便在一定条件下，“i<101”为假，从而退出循环。

程序运行结果如下：

```
1+2+3+...+100=5050
```

【例 4-13】 程序 4-13：while 语句，求 100 以内的奇数、偶数之和。

```
#01: //程序 4-13
#02: #include <stdio.h>
#03: int main(){
#04:     int i,odd,even;
#05:
#06:     i=1;
#07:     odd=0;
#08:     even=0;
#09:
#10:     while (i<101){
#11:         if (i%2==0)
#12:             even+=i;
#13:         else
#14:             odd+=i;
#15:         i++;
#16:     }
#17:
#18:     printf("1+3+...+99=%d,2+4+...+100=%d\n",odd,even);
#19:     return 0;
#20: }
```

程序解释：

#07, #08：奇数之和存放在变量 odd 中，偶数之和存放在变量 even 中。

#10：i 作为循环变量，当小于 101 时执行循环体。

#11～#14：在循环体内有一个 if-else 语句，当 i 为偶数时，even 加 i；当 i 为奇数时，odd 加 i。

程序运行结果如下：

```
1+3+...+99=2500,2+4+...+100=2550
```

2．do-while 语句

do-while 语句的格式为：

```
do
    循环体语句
while (表达式)
```

其含义是：先执行循环体中的语句，然后再判断表达式（循环条件）是否为真，如果表达式为真，

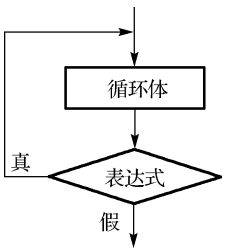


图 4-9 do-while 语句流程图

则继续循环；如果表达式为假，则终止循环。其过程如图 4-9 所示。

对于 do-while 语句应注意：

（1）do-while 语句先执行循环体，后判断表达式，因此至少会执行一次循环语句；

（2）do-while 语句可转化成 while 语句。

【例 4-14】 程序 4-14：do-while 语句，求 1+2+3+…+100 的值。

```
#01: //程序 4-14
#02: #include <stdio.h>
#03: int main(){
#04:     int i,sum;
#05:
#06:     i=1;
#07:     sum=0;
#08:
#09:     do{
#10:         sum+=i;
#11:         i++;
#12:     }while (i<101);
#13:
#14:     printf("1+2+3+...+100=%d\n",sum);
#15:     return 0;
#16: }
```

程序解释：

#09～#11：先执行循环体，即使 i 不满足条件。

#12：判断条件“i<101”是否成立，如果为假，则退出循环。注意在 while (i<101)后面有一个分号“;”，表示 do-while 语句结束。

程序运行结果如下：

```
1+2+3+...+100=5050
```

4.3.2 for 语句

for 语句比较灵活，可以取代 while 语句和 do-while 语句，for 语句的格式为：

for(表达式 1; 表达式 2; 表达式 3)
    循环体语句;

其含义是：①求值表达式 1；②判断表达式 2 是否为真，若表达式为真，则执行循环体语句，若表达式为假，则结束循环；③求值表达式 3；④转到步骤②执行，以决定是否继续循环。其过程如图 4-10 所示。

for 语句可表示为如下简单易理解的格式：

for (循环变量赋初值; 循环条件; 改变循环变量值)
    循环体语句;

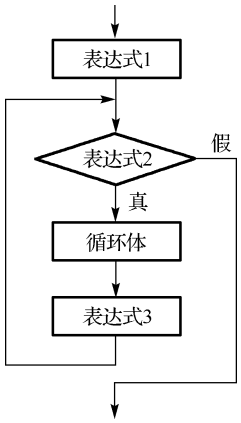


图 4-10　for 语句流程图

“循环变量赋初值”总是一条赋值语句，用来给循环控制变量赋初值；“循环条件”是一个关系表达式或逻辑表达式，决定执行循环体还是结束循环；“改变循环变量值”用来定义循环控制变量每循环一次后按什么方式变化。

【例 4-15】 程序 4-15：for 语句，求 1+2+3+…+100 的值。

```
#01: //程序 4-15
#02: #include <stdio.h>
#03: int main(){
#04:     int i,sum;
#05:
#06:     sum=0;
#07:     for(i=1;i<101;i++)
#08:         sum+=i;
#09:
#10:     printf("1+2+3+...+100=%d\n",sum);
#11:     return 0;
#12: }
```

程序解释：

#07,#08：for 语句中表达式 1 是对循环变量 i 赋初值 1 ，表达式 2 是判断条件“i<101”是否成立，表达式 3 是修改循环变量 i 的值。

程序运行结果如下：

```
1+2+3+...+100=5050
```

对于 for 语句应注意以下事项。

（1）for 语句可以转换成 while 语句或 do-while 语句，例如：

```
表达式 1;
while (表达式 2){
    循环体语句;
    表达式 3;
}
```

（2）for 语句中的表达式 1、表达式 2、表达式 3 的类型任意，都可以省略，但分号“;”不可省略。

① 可以省略表达式 1（循环变量赋初值），表示不对循环控制变量赋初值。此时必须在 for 语句之前给循环变量赋初值。

② 一般不省略表达式 2（循环条件），省略时如果不做其他处理，便成为无限循环。

③ 可以省略表达式 3（改变循环变量值），但必须在循环体中修改循环变量，以使条件表达式在某种情况下会变为假而结束循环；或者通过 break 等语句来终止循环。

④ 若同时省略表达式 1 和表达式 3，则相当于 while 语句。如“for (; i<101;) {sum+=i; i++;}”等价于“while (i<101) {sum+=i; i++;}”。

⑤ 三个表达式均省略，即“for(; ;)”为无限循环，相当于“while(1)”。

（3）表达式 1 和表达式 3 可以是逗号表达式，以使循环变量值在修改时可以对其他变量进行赋值。如“for (sum=0, i=1; i<=101; i++, i++)”等价于“sum=0; for (i=1; i<101; i+=2)”。

（4）表达式 2（循环条件）可以是关系表达式或逻辑表达式，也可以是数值表达式或字符表达式，只要表达式值为真，执行循环体语句。如“for (i=0; (c=getchar( ))!='\n'; i+=c);”。

【例 4-16】 程序 4-16：for 语句，输出字母 a～z。

```
#01: //程序 4-16
#02: #include <stdio.h>
#03: int main(){
#04:     int i;
#05:
#06:     i=0;
#07:     for(;i<26;i++)
#08:         putchar('a'+i);
#09:     printf("\n");
#10:
#11:     i=0;
#12:     for(;i<26;)
#13:         putchar('a'+i++);
#14:     printf("\n");
#15:
#16:     for(i=0;i<26;putchar('a'+i),i++)
#17:         ;
#18:     return 0;
#19: }
```

程序解释：

#06～#08：省略 for 语句中的表达式 1。

#11～#13：省略 for 语句中的表达式 1 和表达式 3。

#16～#17：表达式 3 为一个逗号表达式，此时循环体没有内容，用“;”表示空语句。

程序运行结果如下：

```
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyz
```

4.3.3　循环语句嵌套

while 语句、do-while 语句、for 语句都可以用来处理同一个问题，一般可以互相代替。while 语句和 do-while 语句的循环体中应包括使循环趋于结束的语句。for 语句功能最强。用 while 语句和 do-while 语句时，循环变量初始化的操作应在 while 和 do-while 语句之前完成，而 for 语句可以在表达式 1 中实现循环变量的初始化。

在这三种循环语句中，一个循环体内可以包含另一个完整的循环结构，构成循环的嵌套。应注意以下事项。

（1）三种循环语句可以互相嵌套，层数不限。

（2）外层循环语句可包含两个以上的内层循环语句，但不能相互交叉。例如，以下都是合法的嵌套循环形式。

```
while(){
    while(){
    }
}
```

```
while(){
    do{
    }while( );
}
```

```
for( ; ;) {
    do{
    }while();
    while(){
    }
}
```

（3）嵌套循环内层循环和外层循环的循环控制变量不能相同，同层循环的循环控制变量可以相同。

（4）内层循环的循环控制变量变化较快，外层循环的循环控制变量变化较慢。

【例 4-17】 程序 4-17：循环语句嵌套，输出 1～9 的乘法表。

```
#01: //程序 4-17
#02: #include <stdio.h>
#03: int main(){
#04:     int i,j;
#05:
#06:     for(i=1;i<10;i++){
#07:         for(j=1;j<i+1;j++)
#08:             printf("%d*%d=%2d ",i,j,i*j);
#09:         printf("\n");
#10:     }
#11:     return 0;
#12: }
```

程序解释：

#06：外层循环，循环控制变量为 i，控制乘法表的行数（9 行）。

#07：内层循环，循环控制变量为 j，控制每一行的项数，因此 j 从 1 开始，到 i 值结束循环。

#08：内层循环语句，输出乘法表一行。

#09：外层循环语句，在#07～#08 执行完后执行一次，对乘法表换行。

程序运行结果如下：

```
1*1= 1
2*1= 2 2*2= 4
3*1= 3 3*2= 6 3*3= 9
4*1= 4 4*2= 8 4*3=12 4*4=16
5*1= 5 5*2=10 5*3=15 5*4=20 5*5=25
6*1= 6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1= 7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1= 8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1= 9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

4.3.4 break 与 continue 语句

1．break 语句

break 语句用在循环语句和 switch 语句中，用于终止并跳出循环体或 switch 语句体。当 break 用于 switch 语句中时，可使程序不执行 switch 剩余的语句而跳出 switch 语句体。

当 break 语句用于 while、do-while、for 语句的循环体中时，可使程序终止循环，不再执行循环体内的语句。通常 break 语句与 if 语句联合使用，当满足条件时，便跳出循环体。

对于 break 语句应注意：

（1）break 语句不能用于循环语句和 switch 语句之外的任何语句中；

（2）在多层循环中，一个 break 语句只能向外跳出一层循环体，即 break 语句只能终止并跳出最近一层包含该 break 语句的循环体。

【例 4-18】 程序 4-18：break 语句。

```
#01: //程序 4-18
#02: #include <stdio.h>
#03: int main(){
#04:     char ch;
#05:
#06:     while(1){
#07:         if ((ch=getchar())=='.')
#08:             break;
#09:         putchar(ch);
#10:     }
#11:
#12:     return 0;
#13: }
```

程序解释：

#06：while(1)表示无限循环。

#07, #08：如果输入一个点“.”，则跳出 while 循环。

#09：如果输入不是一个点“.”，则将其输出。

```
abcdefg
abcdefg
1234567890
1234567890
.
```

2．continue 语句

continue 语句会结束本次循环，跳过循环体中尚未执行的语句，进行下一次是否执行循环体的条件判断。

continue 语句只能用在 while、do-while、for 语句的循环体中，通常 continue 语句与 if 语句联合使用，当满足条件时，跳过循环体中的剩余语句而强行执行下一次循环。

【例 4-19】 程序 4-19：continue 语句。

```
#01: //程序 4-19
#02: #include <stdio.h>
#03: int main(){
#04:     int i;
#05:
#06:     for(i=1;i<101;i++){
#07:         if(i%3!=0)
#08:             continue;
#09:         printf("%3d",i);
#10:     }
#11:
#12:     return 0;
#13: }
```

程序解释：

#07, #08：当 i 不是 3 的倍数时，执行 continue 语句而跳过循环体其他语句（#09 行），转向 for 语句的“i++”处执行，并进行下一次循环条件的判断。

```
  3  6  9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78
81 84 87 90 93 96 99
```

4.3.5　循环结构程序设计

循环结构是程序设计中最能发挥计算机特长的结构，可以避免源程序的重复书写，用来描述重复执行某段算法的问题。

循环结构的三个要素是：循环变量、循环体和循环终止条件。当循环条件成立时，执行循环体，当条件不成立时，跳出循环，执行循环结构后面的代码。循环体应包含趋于结束的语句，即修改循环变量的值，否则可能形成一个无限循环。

【例 4-20】 程序 4-20：已知公式：$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 求π的近似值，直到第 n 项的绝对值小于 10^{-8} 为止。

分析该程序的流程，程序中要求对公式中的每一项进行累加，公式的通项公式为：$a_n=(-1)^{n-1}\frac{1}{2n-1}$，用变量 item 表示，因此使用循环语句进行累加求和。

当 $\frac{1}{2n-1}<10^{-8}$ 时，即 $\text{fabs}(a_n)<10^{-8}$ 时，循环终止。

从而，我们要实现的程序的流程图如图 4-11 所示。

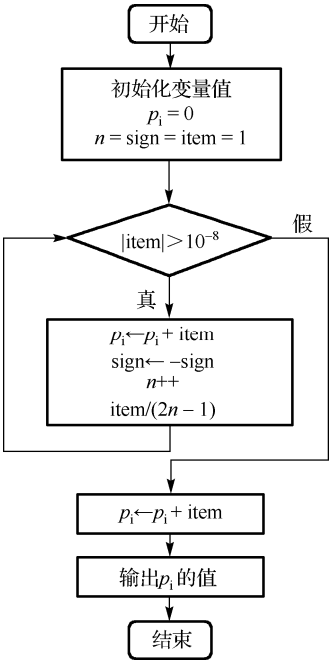


图 4-11　计算 π 的流程图

程序如下：

```
#01: //程序 4-20
#02: #include <stdio.h>
#03: #include <math.h>
#04: int main(){
#05:     double pi,item;
#06:     int n,sign;
#07:
#08:     pi=0;
#09:     n=1;
#10:     sign=1;
#11:     item=1;
#12:
#13:     while(fabs(item)>=1e-8){
#14:         pi+=item;
#15:         sign=-sign;
#16:         n++;
#17:         item=1.0*sign/(2*n-1);
#18:     }
#19:
#20:     pi*=4;
#21:     printf("pi=%10.8lf\n",pi);
```

```
    return 0;
}
```

程序解释：

#08～#11：初始化变量值，其中 sign 作为每项的“+/–”号，item 为通项公式。

#13：fabs 函数是对浮点数求绝对值。

#17：计算新的一项，因为 sign 和(2n–1)都是整数，因此前面要加“1.0”，否则计算结果为整数 0。

```
pi=3.14159263
```

4.4　程 序 示 例

【例 4-21】 程序 4-21：输入三个整数，按从大到小的顺序输出。

```
//程序 4-21
#include <stdio.h>
int main(){
    int x,y,z,temp;

    printf("Enter 3 integers:");
    scanf("%d %d %d",&x,&y,&z);

    if (x<y){
        temp=x;
        x=y;
        y=temp;
    }

    if (x<z){
        temp=x;
        x=z;
        z=temp;
    }

    if (y<z){
        temp=y;
        y=z;
        z=temp;
    }

    printf("Descending order:%d,%d,%d\n",x,y,z);
    return 0;
}
```

程序解释：

#09～#13：若 x<y，则交换 x 和 y，使得 x 和 y 按照从大到小排序。

#15～#19：若 x<z，则交换 x 和 z，使得 x 和 z 按照从大到小排序，从而 x 在三者中最大。

#21～#25：若 y<z，则交换 y 和 z，使得 y 和 z 按照从大到小排序，从而 z 在三者中最小。
程序运行结果如下：

```
Enter 3 integers:2 1 3
Descending order:3,2,1
```

【例 4-22】 程序 4-22：计算器程序，用户输入运算数和四则运算符，输出计算结果。

```
#01: //程序 4-22
#02: #include <stdio.h>
#03: int main(){
#04:     double x,y,result;
#05:     char op;
#06:
#07:     printf("Enter an expression: x+[-*/]y:");
#08:     scanf("%lf%c%lf",&x,&op,&y);
#09:
#10:     switch(op){
#11:         case '+':
#12:             result=x+y;
#13:             break;
#14:         case '-':
#15:             result=x-y;
#16:             break;
#17:         case '*':
#18:             result=x*y;
#19:             break;
#20:         case '/':
#21:             result=x/y;
#22:             break;
#23:         default:
#24:             printf("Input Error!\n");
#25:             return -1;
#26:     }
#27:
#28:     printf("%lf%c%lf=%lf\n",x,op,y,result);
#29:     return 0;
#30: }
```

程序解释：
#11～#13：如果输入的是加号“+”，则进行加法运算。
#14～#16：如果输入的是减号“–”，则进行减法运算。
#17～#19：如果输入的是乘号“*”，则进行乘法运算。
#20～#22：如果输入的是除号“/”，则进行除法运算。
#23～#25：当输入运算符不是“+”、“–”、“*”、“/”之一时，给出错误提示并退出程序。
程序运行结果如下：

```
Enter an expression: x+[-*/]y:9.9*99.9
9.900000*99.900000=989.010000
```

【例 4-23】 程序 4-23：计算并输出 Fibonacci 数列的前 20 项。

斐波那契（Fibonacci）数列，又称黄金分割数列，在现代物理、化学等领域都有直接的应用，如松果、树叶的排列、向日葵的花瓣数、蜂巢、蜻蜓翅膀等都有斐波那契数。为此美国数学会从 1963 起出版了以《斐波那契数列季刊》为名的期刊，专门刊载这方面的研究成果。

Fibonacci 数列的通项公式为：

$$F_n=\begin{cases}1, & n=1\\ 1, & n=2\\ F_{n-1}+F_{n-2}, & n\geqslant 3\end{cases}$$

```
#01: //程序 4-23
#02: #include <stdio.h>
#03: #include <math.h>
#04: int main(){
#05:     int n;
#06:     long int fn_1,fn_2,fn;
#07:
#08:     fn_1=1;
#09:     fn_2=1;
#10:     printf("%10ld%10ld",fn_1,fn_2);
#11:
#12:     for(n=3;n<41;n++){
#13:         fn=fn_1+fn_2;
#14:         printf("%10ld",fn);
#15:         if (n%5==0)
#16:             printf("\n");
#17:         fn_2=fn_1;
#18:         fn_1=fn;
#19:     }
#20:     return 0;
#21: }
```

程序解释：

#08～#10：初始化数列第 1、2 项，并输出。

#13～#14：计算第 n 项，并输出。

#15～#16：控制每行输出 5 项。

#17～#18：重新调整 n–2 和 n–1 项的值。

程序运行结果如下：

```
         1         1         2         3         5
         8        13        21        34        55
        89       144       233       377       610
       987      1597      2584      4181      6765
     10946     17711     28657     46368     75025
    121393    196418    317811    514229    832040
   1346269   2178309   3524578   5702887   9227465
  14930352  24157817  39088169  63245986 102334155
```

【例 4-24】 程序 4-24：输入一个数，判断是否为素数，然后循环输入并判断，直至输入字符“n”结束。

```
#01: //程序 4-24
#02: #include <stdio.h>
#03: #include <math.h>
#04: int main(){
#05:     int i,k,num;
#06:     char answer;
#07:
#08:     do{
#09:         printf("Input a integer:");
#10:         scanf("%d",&num);
#11:
#12:         k=sqrt(num);
#13:         for(i=2;i<k+1;i++)
#14:             if(num%i==0)
#15:                 break;
#16:
#17:         if(i==k+1)
#18:             printf("%d is a prime number.\n",num);
#19:         else
#20:             printf("%d is NOT a prime number.\n",num);
#21:
#22:         fflush(stdin);
#23:         printf("Input another number?(y/n):");
#24:         answer = getchar();
#25:     }while(answer!='n');
#26:
#27:     return 0;
#28: }
```

程序解释：

#12：sqrt( )函数是求平方根，判断一个数的因子，只需判断到该数的平方根。

#13～#15：如果 num 能被 i 整除，则 i 为 num 的一个因子，num 不是素数，跳出循环。

#17：如果是 break 语句导致跳出循环，则此时 i<k+1，否则是#13 行 for 语句循环完，num 没有任何因子 i，此时 i==k+1。

#22～#25：清除输入缓冲区，并读取字符决定是否继续循环。

程序运行结果如下：

```
Input a integer:51
51 is NOT a prime number.
Input another number?(y/n):y
Input a integer:103
103 is a prime number.
Input another number?(y/n):n
```

上机实验：控制结构程序设计应用

本次实验掌握 C 语言程序的两种控制结构：选择结构和循环结构，掌握关系运算符和逻辑运算符，熟练掌握 if 语句、for 语句等控制语句。

（1）有一函数：

$$y=\begin{cases}-x, & x<0\\ 0, & x=0\\ x, & x>0\end{cases}$$

编写程序，输入一个 x 值，输出 y 值。

程序示例：

```
#include <stdio.h>
int main(){
   float x,y;

   printf("Input x:");
   scanf("%f",&x);

   if (x<0)
      y=-x;
   else if (x==0)
      y=0;
   else
      y=x;

   printf("x=%f,y=%f\n",x,y);
   return 0;
}
```

（2）分析比较下列 while 和 do-while 语句，输入不同的值（1、10、100、101），观察输出结果，分析原因。

```
//1. while
#include <stdio.h>
int main(){
   int sum=0,i;

   printf("Input a number:");
   scanf("%d",&i);

   while(i<101){
      sum+=i;
      i++;
   }
```

```
    printf("sum=%d\n",sum);
    return 0;
}
```

```
//2. do-while
#include <stdio.h>
int main(){
    int sum=0,i;

    printf("Input a number:");
    scanf("%d",&i);

    do{
        sum+=i;
        i++;
    }while(i<101);

    printf("sum=%d\n",sum);
    return 0;
}
```

（3）输入两个整数 a、b，求 $\sum_{i=a}^{b} i$。

示例程序：

```
#include <stdio.h>
int main(){
    int i,j,a,b,sum=0;

    printf("Input two number:");
    scanf("%d %d",&a,&b);

    if(a>b){
        i=b;
        j=a;
    }else{
        i=a;
        j=b;
    }

    for(;i<j+1;i++)
       sum+=i;

    printf("sum=%d",sum);
    return 0;
}
```

（4）从键盘输入一个正整数，将该数前后倒置后输出。

示例程序：

```
#include <stdio.h>
int main(){
    int num, right_digit;

    printf("Enter the number:");
    scanf("%d",&num);
    printf("The number in reverse order is:");

    do{
        right_digit=num%10;
        printf("%d",right_digit);
        num/=10;
    } while (num!=0);

    return 0;
}
```

习　题

1．从键盘输入三个整数，求出最大数并输出。

2．从键盘输入一个字符，若为小写字母，则转化为大写字母；若为大写字母，则转化为小写字母，否则保持不变。

3．从键盘输入一个百分制的整数成绩，将其转化为等级分数并输出。90 分以上等级为 A，80～90 分等级为 B，70～80 分等级为 C，60～70 分等级为 D，60 分以下等级为 E。

4．已知方程 $ax^2+bx+c=0$ 的系数值（设 $a\neq0$），求方程的根。

5．编写程序，从键盘上输入一行字符，并依次显示在屏幕上。

6．求 1!+2!+3!+4!+5!+6!+7!+8! 之和。

7．求 1+1/3+1/5+…+1/99 之和。

8．已知序列 1/2, 2/3, 3/5, 5/8,…求其前 100 项之和。

9．输入一个年份，判断是否为闰年。

10．输入一个整数，输出它的所有因子。

11．求 1000 至 10000 之间的所有素数。

12．“水仙花数”是一个三位数，其各位数的立方和等于该数本身。编程打印所有的水仙花数。

13．一个数如果恰好等于它的因子之和，这个数就称为完数。求 100 之内的所有完数。

14．读入一系列整数，统计出正整数的个数和负整数的个数，读入 0 则结束。

15．从键盘输入一批字符（以@结束），按要求加密并输出。

加密规则：①所有字母均转换为小写；②若是字母 'a'～'y'，则转化为下一个字母；③若是 'z'，则转化为 'a'；④其他字符，保持不变。

第 5 章　数组和字符串

数组是按顺序排列的具有相同类型的变量序列。在编写程序过程中，有时需要定义多个相同类型的变量，为了简化程序设计，可以把这些具有相同类型的若干变量按有序的形式组织起来，形成数组。

一个数组包含一个或多个数组元素，这些数组元素具有相同的数据类型。按数组元素的数据类型种类，数组可分为数值数组、字符数组、指针数组、结构数组等。本章介绍数值数组和字符数组（字符串）。

可以用数组名标识数组，用数组名和下标确定数组中的元素，数组元素相当于简单变量。

5.1　一 维 数 组

5.1.1　一维数组定义

1．定义方式

一维数组定义方式为：“数据类型 数组名[常量表达式];”。其中“数据类型”是任意一种基本数据类型（如 int、double 等）或其他类型，“数组名”是用户定义的数组标识符，方括号中的“常量表达式”表示数据元素的个数，也称为数组长度。如“float temperature[24]”、“char buffer[80]”、“float x[5+5]”都是合法的数组定义。

对于数组应注意以下事项。

（1）数组的类型是指数组元素的取值类型，同一个数组的所有元素的数据类型都是相同的。

（2）可以在同一个类型说明中定义多个数组和多个变量。

例如，“float x[5], y[8+2];”表示同时定义两个实型数组，一个是数组 x，有 5 个元素，另一个是数组 y，有 10 个元素。

又如，“int a,b,c,num[5],sum[5];”表示同时定义了三个整型变量 a、b、c 和两个整型数组 num、sum。

（3）数组名应是合法的标识符，不能与变量名或其他标识符相同。例如，“int price;”，如果又定义“double price[5];”则是错误的。

（4）方括号中的元素个数定义可以是符号常量或常量表达式，但不能是变量。

例如，如果定义了一个常量：“#define LEN 10”，然后定义“char buff[2*LEN+1];”表示定义了一个字符类型的数组 buff，大小为 21。

如果先定义了一个变量：“int len=10;”，然后定义数组“char buff[len];”是非法的，因为不能用变量定义数组长度，只能使用“常量表达式”。

（5）方括号中的“常量表达式”表示数组的元素总数，但是数组元素的下标从 0 开始。如“int color[3];”表示定义一个整型类型的 color 数组，共有三个元素，分别为 color[0]、color[1]和 color[2]。

2．存储方式

一维数组在存储时，用连续的内存单元存放数组的各个元素。

如定义“int a[5];”，则在内存中的存储示意图如图 5-1 所示。内存以字节为单元组成，每个字节都

有一个地址，图中形如“0x28ff0c”的为内存的地址。数组名“a”代表数组第一个元素“a[0]”在内存中的地址（简称数组首地址），是一个地址常量。在 32 位计算机，一个“int”类型变量分配 4 字节的内存大小，如数组元素“a[0]”分配了地址为 0x28ff0c～0x28ff0f 的 4 字节。

一维数组在编译时被分配连续的内存空间，分配的内存总字节数=数组长度*sizeof(元素数据类型)。例如，数组“int a[5];”的每个元素分配 4 字节（sizeof(int)=4），则总共分配了 4×5=20 字节的内存空间。

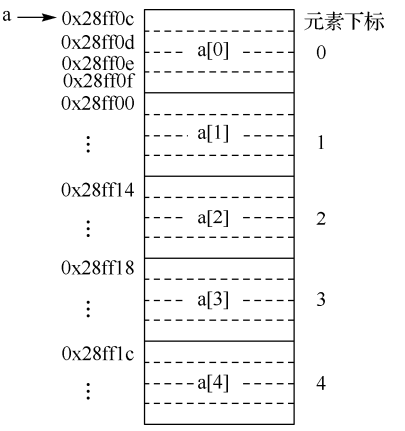


图 5-1　一维数组 a[5]存储示意图

3．初始化

在定义数组时为数组元素赋初值，称为数组的初始化。数组初始化在编译阶段进行，因此可以减少运行时间，提高效率。初始化的一般形式为：“数据类型 数组名[常量表达式]={数值, 数值,…, 数值};”，在花括号“{}”中的数值为对应的数组元素的初值，之间用逗号间隔。

例如，“int a[5]={1,2,3,4,5};”等价于“a[0]=1; a[1]=2; a[2]=3; a[3]=4; a[4]=5;”。

数组初始化应注意以下事项。

（1）如果不对数组初始化，其元素初始值为所在内存单元原来的值，是不可预知的，因此数组元素的值为随机值。

（2）定义数组时可以对全部数组元素赋初值，此时在数组定义时可以不指定数组长度，即保持方括号“[]”为空，编译器根据初值个数确定数组的长度。如“int a[5]={1,2,3,4,5};”等价于“int a[]={1,2,3,4,5};”。

（3）定义数组时可以只对部分元素赋初值，当“{}”中数值个数少于数组元素个数时，只对数组前面部分元素赋初值，此时后面剩余元素的初值为 0。如“int a[5]={1,2,3};” 等价于“a[0]=1; a[1]=2; a[2]=3; a[3]=0; a[4]=0;”。

（4）定义数组时不能对不连续的部分元素或后面的连续元素赋初值。如“int a[5]={ , ,3,4,5};”、“int a[5]={1, ,3, ,5};”都是错误的。

（5）定义数组时，初始值的个数不能大于数组元素的个数。如“int a[5]={1,2,3,4,5,6};”是错误的。

（6）只能给数组元素逐个赋初值，不能给数组整体赋初值。如“int a[5]=1;”是错误的。

5.1.2　一维数组元素引用

数组元素是组成数组的基本单元，必须先定义数组，然后才能使用数组元素。引用数组元素的一般形式为“数组名[下标]”，其中“下标”可以是整型常量或整型表达式，表示元素在数组中的顺序号。例如，如果定义“int a[5];”，则“a[3]”、“a[i]”、“a[i+j]”等都是合法的数组元素引用方式。

数组元素是一种变量，也称为下标变量。只能逐个引用数组元素，不能一次引用整个数组。例如，如果要输出数组 a[5]的所有元素，“printf("%d",a);”是错误的，必须使用循环语句逐个输出各数组元素：

```
for(i=0;i<5;i++)
    printf("%d",a[i]);
```

【例 5-1】 程序 5-1：输入 5 个成绩，然后按输入次序相反的顺序输出。

```
#01: //程序 5-1
#02: #include <stdio.h>
#03: #define LEN 5
#04: int main(){
#05:    int i;
#06:    float score[LEN];
#07:
#08:    printf("Enter %d scores:",LEN);
#09:    for(i=0;i<LEN;i++)
#10:       scanf("%f",&score[i]);
#11:
#12:    printf("The scores in reverse order are:");
#13:    for(i=LEN-1;i>-1;i--)
#14:       printf("%3.1f ", score[i]);
#15:
#16:    return 0;
#17: }
```

程序解释：

#03：定义数组长度为 5。

#09, #10：使用 for 循环输入数组的所有元素。

#13, #14：使用 for 循环将数组的所有元素倒序输出。

程序运行结果如下：

```
Enter 5 scores:85.5 98.2 95 77.8 82
The scores in reverse order are:82.0 77.8 95.0 98.2 85.5
```

通常使用循环语句和 scanf( ) 函数对数组元素赋值，使用循环语句和 printf( ) 函数输出数组元素。

【例 5-2】 程序 5-2：输入 6 个数，输出其平均值。

```
#01: //程序 5-2
#02: #include <stdio.h>
#03: #define LEN 6
#04: int main(){
#05:    int i;
#06:    float average,num[LEN];
#07:
#08:    printf("Enter %d numbers:",LEN);
#09:    for(i=0;i<LEN;i++)
#10:       scanf("%f",&num[i]);
#11:
#12:    average=0;
#13:    for(i=0;i<LEN;i++)
#14:       average+=num[i];
#15:
#16:    average/=LEN;
#17:    printf("Average=%4.1f\n",average);
#18:    return 0;
#19: }
```

程序解释：

#09, #10：使用 for 语句逐个输入 6 个数。

#12～#14：计算 6 个数的总和。

#16：计算平均值。

程序运行结果如下：

```
Enter 6 numbers:1.1 2.2 3.3 4.4 5.5 6.6
Average= 3.9
```

5.2　二 维 数 组

5.2.1　二维数组定义

1．定义方式

一维数组只有一个下标，其数组元素也为单下标变量。*N* 维数组有 *N* 个下标，其数组元素为 *N* 下标变量，*N* 个下标用来标识它在数组中的位置。本节重点介绍二维数组，*N* 维数组可以类推得到。

二维数组的一般形式为：“数据类型 数组名[常量表达式 1][常量表达式 2]”，其中“常量表达式 1”表示第一维下标的长度（行数），“常量表达式 2”表示第二维下标的长度（列数）。二维数组总的元素个数等于常量表达式 1 乘以常量表达式 2。如“int a[3][4];”、“float score[10][3];”、“char name[3][5];”等都是合法的二维数组定义。

二维数组可以理解为一个一维数组，每个数组元素又是一个一维数组。例如“int a[2][3];”定义一个 2 行 3 列共计 6 个元素的整型数组，如图 5-2 所示。从图 5-2 可以看出，二维数组 a 是由两个元素 a[0]、a[1]组成的，每个元素 a[i]又包含一个三个元素的一维数组，即 a[0]包含 a[0][0]、a[0][1]、a[0][2]；a[1]包含 a[1][0]、a[1][1]、a[1][2]。

2．存储方式

二维数组的下标在行和列两个方向上变化，下标变量在二维数组中的位置处于一个平面中，但是内存是一维的，即内存地址是按一维线性排列的。为了在内存中存储二维数组，采取“按行排列”的排列顺序，即存放完二维数组中的一行元素后，再顺次存放二维数组的第二行……以此类推。

例如，“int a[2][3];”在内存中的存储示意图如图 5-3 所示。

图 5-2　二维数组 a[2][3]

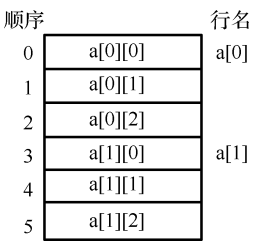


图 5-3　二维数组 a[2][3]存储示意图

先存放 a[0]行、再存放 a[1]行，每行中的三个元素也是依次存放。每个元素占 4 字节的内存空间。二维数组共占 4×6=24 字节的数据。

从数组元素的存放顺序可以看出，二维数组的右边下标比左边下标变化快。对于 *N* 维数组（空间矩阵），最右下标变化最快。

3．初始化

在定义二维数组时为各下标变量赋初值，称为二维数组的初始化，初始化的方法有两种：①按行分段初始化；②按元素排列顺序初始化。

（1）按行分段初始化

对二维数组所有元素全部初始化，例如，“int a[2][3]={{1,2,3},{4,5,6}};”对二维数组每个元素均赋初值，如图 5-4(a)所示。

对二维数组部分元素初始化，此时没有赋初始值的元素值为“0”，例如，“int a[2][3]={{1,2},{4}};”，使得“a[0][0]=1、a[0][1]=2、a[1][0]=4”，其他元素为 0，如图 5-4(b)所示。

按行分段初始化时，可以省略第一维长度值。例如，“int a[][3]={{1},{4,5}};”等价于“int a[2][3]={{1},{4,5}};”，如图 5-4(c)所示。

（2）按元素排列顺序初始化

对二维数组所有元素全部初始化，例如，“int a[2][3]={1,2,3,4,5,6};”，如图 5-4(d)所示。

对二维数组部分元素初始化，此时只能为数组前面部分元素赋初值，后面剩余元素的初值为“0”。例如，“int a[2][3]={1,2,4};”，使得“a[0][0]=1、a[0][1]=2、a[0][2]=4”，其他元素为 0，如图 5-4(e)所示。

按元素排列顺序初始化时，可以省略第一维长度值，编译器根据列数自动计算行数。例如，“int a[][3]={1,2,3,4,5};”等价于“int a[2][3]={1,2,3,4,5};”，如图 5-4(f)所示。

	a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]
(a)	1	2	3	4	5	6
(b)	1	2	0	4	0	0
(c)	1	0	0	4	5	0
(d)	1	2	3	4	5	6
(e)	1	2	4	0	0	0
(f)	1	2	3	4	5	0

图 5-4　二维数组初始化示意图

5.2.2　二维数组元素引用

二维数组的元素为双下标变量，形式为：“数组名[下标 1][下标 2]”，其中“下标 1”标识行位置，“下标 2”标识列位置，可为整型常量或整型表达式。如“a[2][3]”表示二维数组第 2 行第 3 列（均从 0 开始计算）位置处的元素。

数组定义时的下标和数组元素引用的下标有如下不同。

（1）数组定义时方括号中的下标是某一维的长度，即该维元素的个数；而数组元素引用下标是该元素在数组中的位置标识，从 0 开始计算位置。

（2）数组定义时的下标必须是常量或常量表达式；数组元素引用时的下标可以是常量、变量或表达式。

（3）数组元素引用的下标最大值不能超过定义时的长度。如定义二维数组“int a[4][5];”，行数最大为 3，列数最大为 4，即最大数组元素为 a[3][4]。

【例 5-3】 程序 5-3：将二维数组行、列元素互换，存到另一个数组中。

```
#01: //程序 5-3
#02: #include <stdio.h>
#03: #define ROW 3
#04: #define COL 4
#05: int main(){
#06:     int a[ROW][COL]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
#07:     int aT[COL][ROW],i,j;
#08:
#09:     printf("array a:\n");
#10:     for(i=0;i<ROW;i++){
#11:         for(j=0;j<COL;j++)
#12:             printf("%5d",a[i][j]);
#13:         printf("\n");
#14:     }
#15:
#16:     for(i=0;i<ROW;i++)
#17:         for(j=0;j<COL;j++)
#18:             aT[j][i]=a[i][j];
#19:
#20:     printf("Array b:\n");
#21:     for(i=0;i<COL;i++){
#22:         for(j=0;j<ROW;j++)
#23:             printf("%5d",aT[i][j]);
#24:         printf("\n");
#25:     }
#26:
#27:     return 0;
#28: }
```

程序解释：

#03, #04：定义二维数组的行数和列数。

#10～#14：用嵌套 for 语句输出二维数组元素。外层循环控制逐行处理，内层循环控制逐列处理。

#13：控制二维数组的每行输出完后换行。

#16～#18：将二维数组元素行和列进行互换。

#21～#25：输出二维数组元素。

程序运行结果如下：

```
array a:
    1    2    3    4
    5    6    7    8
    9   10   11   12
Array b:
    1    5    9
    2    6   10
    3    7   11
    4    8   12
```

【例 5-4】 程序 5-4：输入二维数组元素（3 维×4 维），然后对于每行，选择最小的元素组成一个一维数组并输出。

```
#01: //程序 5-4
#02: #include <stdio.h>
#03: #define ROW 3
#04: #define COL 4
#05: int main(){
#06:     float a[ROW][COL],b[ROW],min;
#07:     int i,j;
#08:
#09:     printf("Input %d*%d array:\n",ROW,COL);
#10:     for(i=0;i<ROW;i++)
#11:         for(j=0;j<COL;j++)
#12:             scanf("%f",&a[i][j]);
#13:
#14:     for(i=0;i<ROW;i++){
#15:         min=a[i][0];
#16:         for(j=1;j<COL;j++)
#17:             if(min>a[i][j])
#18:                 min=a[i][j];
#19:         b[i]=min;
#20:     }
#21:
#22:     printf("Array with the min data:");
#23:     for(i=0;i<ROW;i++)
#24:         printf("%5.2f",b[i]);
#25:
#26:     return 0;
#27: }
```

程序解释：

#10～#12：输入二维数组元素。

#14～#20：寻找二维数组每行的最小值，然后赋值给对应的一位数组。内层循环逐列比较数据大小，内层循环结束后 min 即为该行最小的元素。

#23,#24：输出一维数组的元素。

程序运行结果如下：

```
Input 3*4 array:
1.1 2.2 3.3 4.4
5.5 4.4 3.3 2.2
9.9 7.7 6.6 8.8
Array with the min data: 1.10 2.20 6.60
```

5.3 字 符 串

C 语言没有字符串变量，如果要存储字符串，可以将字符串存入字符数组中。字符数组是一个一维数组，类型为字符型，每个数组元素存放字符串中的一个字符。

字符数组首先是一种数组，数组的定义和引用方法都适用于字符数组；字符数组又是一种特殊的数组，有独特的引用方法。

5.3.1　字符数组和字符串

1．字符数组

用来存放字符或字符串的数组称为字符数组，类型为 char，如“char ch[5];”定义了一个长度为 5 的字符数组。

由于字符型和整型可通用，因此可定义一个整型数组来存放字符数据，此时每个数组元素占 2 字节的内存单元。

也可以定义二维或 *N* 维字符数组，如“char name[10][80];”。

对于字符数组，可以采用逐个元素赋值方法进行初始化。如同一般数组一样，对每个数组元素赋值一个字符。如“char ch[5]={ 'H', 'e', 'l', 'l', 'o'};”，如图 5-5(a)所示。这样赋值可以不加字符串结束符'\0'，建议初始化时加上。

当对所有元素都赋初值时，可以省去长度说明，如“char ch[]={ 'H', 'e', 'l', 'l', 'o'};”等价于“char ch[5]={ 'H', 'e', 'l', 'l', 'o'};”。

当初值个数小于数组长度时，剩余的元素均为 '\0'。如“char ch[5]={ 'H', 'e', 'l' };”，则后两个元素的值为 '\0'，如图 5-5(b)所示。

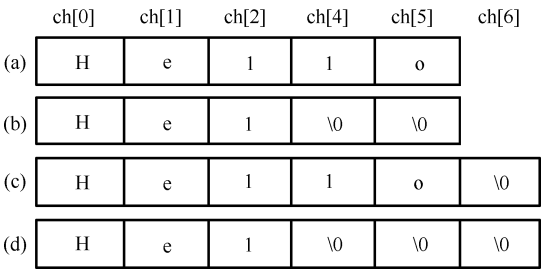


图 5-5　字符数组初始化示意图

可以像普通数组一样，通过下标引用字符数组中的元素。

【例 5-5】 程序 5-5：字符数组示例。

```
#01: //程序 5-5
#02: #include <stdio.h>
#03: #define ROW 5
#04: #define COL 10
#05: int main(){
#06:    char shape[][COL]={{'C','i','r','c','l','e'},
#07:    {'T','r','i','a','n','g','l','e'},
#08:    {'S','q','u','a','r','e'},
#09:    {'R','e','c','t','a','n','g','l','e'},
#10:    {'O','v','a','l'}};
#11:     int i,j;
#12:
#13:     for(i=0;i<ROW;i++){
#14:        for(j=0;j<COL;j++)
```

```
#15:            printf("%c",shape[i][j]);
#16:        printf("\n");
#17:    }
#18:
#19:    return 0;
#20: }
```

程序解释：

#06～#10：初始化二维字符数组，省略了一维下标长度。

#13～#17：逐个字符打印，每行打印一个换行符。

程序运行结果如下：

```
Circle
Triangle
Square
Rectangle
Oval
```

2．字符串

C 语言用一个字符数组来存放一个字符串。字符串以 '\0' 作为结束标志，因此存放字符串时，也把结束符 '\0' 存入字符数组。使用'\0'标志可以很容易确定字符串的结束，此时字符串的长度并不等于字符数组的长度。

可以用字符串常量对字符数组进行初始化。将字符串常量整体赋给数组，如“char ch[6]={"Hello" };”，此时会自动添加字符串结束符 '\0'，如图 5-5(c)所示。

可以省略花括号“{}”，即“char ch[6]="Hello" ;”等价于“char ch[6]={"Hello" };”。

可以不指定数组长度，保持“[]”为空，即“char ch[]="Hello" ;”等价于“char ch[6]="Hello" ;”。

字符串常量初始化方式比逐个元素初始化方法要多占一字节，用于存放字符串结束标志'\0'。用字符串常量赋初值时，必须保证“数组元素个数≥字符个数+1”成立，其中多的一个位置用来存放字符串结束符'\0'。

如果数组元素个数大于字符个数两个以上，剩余位置均被赋值为'\0'。如“char ch[6]="Hel" ;”，后三个元素的值为 '\0'，如图 5-5(d)所示。

对于例 5-5 中，可以修改字符数组定义为:

```
char shape[][COL]={"Circle","Triangle","Square","Rectangle","Oval"};
```

输出结果相同。

3．字符串输入与输出

对于字符串，在程序中既可以逐个引用字符串中的单个字符（字符数组元素），也可以一次引用整个字符串。

（1）逐个字符数组元素引用

此时输入/输出使用“%c”格式符。

如“scanf("%c",&a[i]);”、“printf("%c",a[0]);”等将字符串作为一个个字符进行输入和输出。

（2）整个字符串一次引用

此时输入/输出使用“%s”格式符。

如“scanf("%s",a);”、“printf("%s",a);”等对字符串整体进行输入和输出。

使用“%s”格式符应注意：

（1）使用“%s”进行输入时，参数使用字符数组名，不能加“&”；遇到空格或回车将结束输入；自动在字符串末尾添加“\0”标志。

（2）使用“%s”进行输出时，无论数组元素有多少个，只要遇到“\0”，就结束输出。

（3）“%s”要求的参数是地址。对于一维数组，数组名就是地址；对于二维数组，只写行下标时是地址。

【例 5-6】 程序 5-6：字符串输入/输出示例。

```
#01: //程序 5-6
#02: #include <stdio.h>
#03: #define LEN 20
#04: int main(){
#05:     char str[LEN];
#06:
#07:     printf("Input a string:");
#08:     scanf("%s",str);
#09:     printf("%s",str);
#10:
#11:     return 0;
#12: }
```

程序解释：

#08：scanf( )函数使用“%s”时，遇到空格或回车结束输入。字符数组长度为 20，因此输入字符串的长度必须小于 20，以留出一字节存放字符串结束标志“\0”。

#09：str 中只有“Hello”字符串，将其输出。

程序运行结果如下：

```
Input a string:Hello world
Hello
```

为了避免这种情况，可以定义多个字符数组分段存放含空格的字符串，或使用字符串处理函数。

5.3.2　字符串处理函数

C 语言提供许多字符串处理函数，用来对字符串进行输入、输出、复制、比较等。使用字符串处理函数可以简化程序设计。使用字符串处理函数应包含头文件"string.h"（除了字符串输入和输出函数）。

下面介绍几个经常使用的字符串函数。

1．字符串输入和输出函数

字符串输入函数为“gets( )”，功能是从标准输入设备（键盘）中获取以回车结束的字符串，并自动加“\0”。格式为“gets(字符数组名)”，输入字符串长度应小于字符数组长度。

字符串输出函数为“puts( )”，功能是向标准输出设备（屏幕）输出一个字符串，输出后换行。格式为“puts(字符数组名)”，字符数组必须以“\0”结束。

puts( )和 gets( )函数只能输出或输入一个字符串。

【例 5-7】 程序 5-7：字符串输入/输出函数示例。

```
#01: //程序 5-7
#02: #include <stdio.h>
#03: #define LEN 20
#04: int main(){
#05:     char str[LEN];
#06:
#07:     printf("Input a string:");
#08:     gets(str);
#09:     puts(str);
#10:
#11:     return 0;
#12: }
```

程序解释：

#08：gets( )函数不以空格作为字符串输入结束标志，只以回车作为输入结束标志。

#09：输出整个字符串。

程序运行结果如下：

```
Input a string:Hello world
Hello world
```

2．字符串长度函数

字符串长度函数为“strlen( )”，功能是计算字符串的有效长度，不包括“\0”。格式是“strlen(字符数组名)”。

【例 5-8】 程序 5-8：字符串长度函数示例。

```
#01: //程序 5-8
#02: #include <stdio.h>
#03: #include <string.h>
#04: #define LEN 20
#05: int main(){
#06:     char str[LEN];
#07:
#08:     printf("Input a string:");
#09:     gets(str);
#10:     printf("The string length is %d.\n",strlen(str));
#11:     printf("The char array size is %d.\n",sizeof(str));
#12:
#13:     return 0;
#14: }
```

程序解释：

#10：strlen( )函数返回的字符串长度不包括“\0”。

#11：sizeof( )运算符返回整个字符数组的长度。

程序运行结果如下：

```
Input a string:Hello world
The string length is 11.
The char array size is 20.
```

3．字符串比较函数

不能用“==”对字符串进行比较，必须用字符串比较函数。字符串比较函数为“strcmp( )”，功能是比较两个字符串的大小，将两个字符串从左向右逐个字符地比较它们的 ASCII 码的大小，直至遇到不同字符或“\0”。如果全部字符相同，则相等；否则以第一个不同字符的比较结果为准。

“strcmp( )”函数格式为“strcmp(字符串 1,字符串 2)”，如果字符串 1 等于字符串 2，返回零值；如果字符串 1 大于字符串 2，返回 1 值；如果字符串 1 小于字符串 2，返回–1 值。

也可以用该函数对两个字符串常量、字符数组和字符串常量进行大小比较。

【例 5-9】 程序 5-9：字符串比较函数示例。

```
#01: //程序 5-9
#02: #include <stdio.h>
#03: #include <string.h>
#04: #define LEN 20
#05: int main(){
#06:    char str1[LEN],str2[LEN];
#07:    int result;
#08:
#09:    printf("Input two strings:\n");
#10:    gets(str1);
#11:    gets(str2);
#12:
#13:    result=strcmp(str1,str2);
#14:    if(result==0)
#15:       printf("str1==str2\n");
#16:    else if(result>0)
#17:       printf("str1>str2\n");
#18:    else
#19:       printf("str1<str2\n");
#20:
#21:    return 0;
#22: }
```

程序解释：

#10～#11：输入两个字符串，长度小于 20。

#13：比较两个输入的字符串的大小，将结果保存在变量 result 中。

#14～#19：根据变量 result 的值，可知道比较结果。

程序运行结果如下：

```
Input two strings:
affect
effect
str1<str2
```

4．字符串连接函数

字符串连接函数为“strcat( )”，功能是将两个字符串连接到一起，形成一个新的字符串。格式是“strcat(字符数组名 1,字符数组名 2)”，该函数把字符数组 2 中的字符串连接到字符数组 1 中的字符串的后面，函数返回结果为字符数组 1 的首地址。

使用 strcat( )函数时，字符数组 1 必须足够大，能够容纳两个字符串的内容。连接后，字符串 1 原来的结束标志“\0”被删除，新字符串末尾加“\0”。

【例 5-10】 程序 5-10：字符串连接函数示例。

```
#01: //程序 5-10
#02: #include <stdio.h>
#03: #include <string.h>
#04: #define LEN 20
#05: int main(){
#06:     char str1[LEN],str2[LEN];
#07:
#08:     printf("Input two strings:\n");
#09:     gets(str1);
#10:     gets(str2);
#11:
#12:     strcat(str1,str2);
#13:     puts(str1);
#14:     return 0;
#15: }
```

程序解释：

#12：将输入的两个字符串连接起来，存在 str1 中。

程序运行结果如下：

```
Input two strings:
hello
world
helloworld
```

5．字符串复制函数

字符串复制函数为“strcpy( )”，功能是复制给定的字符串。格式是“strcpy(字符数组名 1,字符串 2)”，该函数将字符串 2 复制到字符数组 1 中，包括字符串结束标志”\0”。字符串 2 可以是字符串常量或字符数组。

使用 strcpy( )函数时，字符数组 1 必须足够大，能够容纳复制的字符串。

必须使用 strcpy( )函数对字符串复制，不能使用赋值语句为一个字符数组赋值。例如，如果定义“char str1[10],str2[10];”，语句“str1=“Hello!”;”和“str2=str1;”都是错误的。

【例 5-11】 程序 5-11：字符串复制函数示例。

```
#01: //程序 5-11
#02: #include <stdio.h>
#03: #include <string.h>
#04: #define LEN 20
#05: int main(){
#06:     char str1[LEN],str2[LEN];
#07:
#08:     printf("Input one strings:\n");
#09:     gets(str2);
```

```
#10:
#11:    strcpy(str1,str2);
#12:    puts(str1);
#13:    return 0;
#14: }
```

程序解释：

#11：将输入的字符串 str2 复制到 str1 中。

程序运行结果如下：

```
Input one strings:
Hello world
Hello world
```

下面是一个综合使用字符串处理函数的例子。

【例 5-12】 程序 5-12：输入 *N* 个字符串，然后按字母顺序输出。

```
#01: //程序 5-12
#02: #include <stdio.h>
#03: #include <string.h>
#04: #define LEN 20
#05: #define N 5
#06: int main(){
#07:    char temp[LEN],strings[N][LEN];
#08:    int i,j,index;
#09:
#10:    printf("Input %d strings:\n",N);
#11:    for(i=0;i<N;i++)
#12:       gets(strings[i]);
#13:
#14:    for(i=0;i<N;i++){
#15:       index=i;
#16:
#17:       for(j=i+1;j<N;j++)
#18:          if(strcmp(strings[index],strings[j])>0)
#19:             index=j;
#20:
#21:       if(index!=i){
#22:          strcpy(temp,strings[i]);
#23:          strcpy(strings[i],strings[index]);
#24:          strcpy(strings[index],temp);
#25:       }
#26:    }
#27:
#28:    printf("Sorted strings:\n",N);
#29:    for(i=0;i<N;i++)
#30:      puts(strings[i]);
```

```
#31:
#32:    return 0;
#33: }
```

程序解释：

#07：使用二维数组存放 *N* 个字符串，每一个一维数组 strings[i]就是一个字符串。

#11～#12：使用 for 语句输入 *N* 个字符串。

#14, #17：这两个 for 语句构成双重循环，完成按字母顺序排序。

在外层 for 循环中把下标 i 赋给 index（变量 index 记录最小字符串的位置）。进入内层 for 循环后，比较 strings[index]与 strings[i]后面的字符串，若有比 strings[index]小者，则把其下标赋给 index。

#21～#25：内层 for 循环结束后，如果 index 不等于 i，说明有比 strings[i]更小的字符串出现，因此交换 strings[i]和 strings[index]的内容。

程序运行结果如下：

```
Input 5 strings:
Miscrosoft
Apple
Google
Facebook
Amazon
Sorted strings:
Amazon
Apple
Facebook
Google
Miscrosoft
```

5.4 程 序 示 例

【例 5-13】 程序 5-13：用冒泡法对 *N* 个整数从小到大进行排序。

冒泡法排序过程如下：

（1）比较第一个数与第二个数，如果“a[0]>a[1]”，则交换两者位置；然后比较第二个数与第三个数；以此类推，直至第 *N*–1 个数和第 *N* 个数比较。完成第一趟冒泡排序，结果最大的数被放置在最后一个元素位置上。

（2）对前 *N*–1 个数进行第二趟冒泡排序，结果使次大的数被放置在第 *N*–1 个元素位置。

（3）重复上述过程，共经过 *N*–1 趟冒泡排序后，排序结束。

```
#01: //程序 5-13
#02: #include <stdio.h>
#03: #define N 5
#04: int main(){
#05:    int i,j,temp,a[N];
#06:
#07:    printf("Input %d numbers:\n",N);
#08:    for(i=0;i<N;i++)
#09:       scanf("%d",&a[i]);
```

```
#10:
#11:    for(i=0;i<N-1; i++)
#12:       for(j=0; j<N-1-i; j++)
#13:           if (a[j]>a[j+1]){
#14:               temp=a[j];
#15:                a[j]=a[j+1];
#16:                a[j+1]=temp;
#17:           }
#18:
#19:    printf("Sorted numbers:\n");
#20:    for(i=0; i<N; i++)
#21:        printf("%4d",a[i]);
#22:
#23:    return 0;
#24: }
```

程序解释：

#07～#09：输入 *N* 个整数。

#11：外层循环控制比较的趟数，共为 *N*–1 趟。

#12：内层循环控制一趟比较的次数，共为 *N*–1–*i* 次。

#13～#17：如果前一个数比后一个数大，则交换它们的位置。

程序运行结果如下：

```
Input 5 numbers:
22 99 11 55 66
Sorted numbers:
  11  22  55  66  99
```

【例 5-14】 程序 5-14：输入一行字符串文本，统计其中有多少个单词（空格隔开）。

在字符串中，分隔单词的空格，一个和多个是等价的。

扫描字符串时，可在两个状态之间转换：一个状态是在单词之内，称为字母状态；一个状态是在单词之外，称为空格状态。为了标志这两种状态，可设一个变量 word：当进入字母状态时，令 word 为 1，且在整个字母状态中保持不变；当进入空格状态时，word 变为 0，且在整个空格状态中保持不变。

为统计单词的个数，设置一个计数器 num，一旦进入字母状态，即在 word 由 0 变为 1 时，该计数器加 1。

```
#01: //程序 5-14
#02: #include <stdio.h>
#03: #define LEN 128
#04: int main(){
#05:    char string[LEN],ch;
#06:    int i,word,num;
#07:
#08:    num=0;
#09:    word=0;
#10:    printf("Input a string:\n");
```

```
#11:    gets(string);
#12:
#13:    for(i=0;(ch=string[i])!='\0';i++)
#14:      if(ch==' ')
#15:         word=0;
#16:      else if(word==0){
#17:         word=1;
#18:         num++;
#19:       }
#20:
#21:    printf("Total %d words.\n",num);
#22:    return 0;
#23: }
```

程序解释：

#14～#15：如果当前字符为空格，则未出现新单词。

#16～#19：如果当前字符不是空格，而前一个字符为空格（word==0），则是新单词开始。

程序运行结果如下：

```
Input a string:
The quick brown fox jumps over the lazy dog
Total 9 words.
```

上机实验：数组程序设计应用

本次实验掌握 C 语言程序的一维、二维数组的定义、赋值和输入/输出等方法。

（1）输入 *N* 个整数，按从小到大的顺序进行排序，要求输入每个整数时，插入已排好序的数组中。

程序示例：

```
#include <stdio.h>
#define N 5
int main(){
   int i,j,k,num,a[N]={0};

   for(i=0;i<N;i++){
      printf("Input a numbers:");
      scanf("%d",&num);

      for(j=0;j<i;j++)
         if(num<a[j]){
            for(k=i;k>j;k--)
               a[k]=a[k-1];
            break;
         }
      a[j]=num;

      printf("Now the numbers in array:\n");
```

```
        for(j=0; j<i+1; j++)
            printf("%4d",a[j]);
        printf("\n");
    }

    return 0;
}
```

（2）打印杨辉三角形（要求打印出 10 行，如下）。

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
……
```

程序示例：

```
#include <stdio.h>
#define ROW 10
#define COL 10
int main(){
    int i,j,a[ROW][COL];

    for(i=0;i<ROW;i++){
        a[i][0]=1;
        a[i][i]=1;
    }

    for(i=2;i<ROW;i++)
        for(j=1;j<i;j++)
            a[i][j]=a[i-1][j-1]+a[i-1][j];

    for(i=0;i<10;i++){
        for(j=0;j<=i;j++)
            printf("%5d",a[i][j]);
        printf("\n");
    }
    return 0;
}
```

习　　题

1．输入 10 个数，输出其中的最大值和最小值。

2．输入 10 个数，将数组中下标为奇数的元素值取倒数后重新存入该数组中，并输出所有元素。

3．输入 10 个数，将数组中右半部分的元素值取相反数后重新存入该数组中，并输出所有元素。

4．青年歌手参加歌曲大奖赛，有 10 个评委对她进行打分，试编程求这位选手的平均得分（去掉一个最高分和一个最低分）。

5．输入一个二维数组（3×4）的元素，输出其中值最大的元素值，以及它的行号和列号。

6．输入一个矩阵（5×5），求矩阵下三角形元素之和。

7．输入一个二维数组（3×4）的元素，在二维数组中选出各行最大的元素组成一个一维数组并输出。

8．一个学习小组有 5 人，每人有三门课的考试成绩。求全组分科的平均成绩和各科总平均成绩。

9．已知两个矩阵 a[3][2]={1,3,5,2,4,6}，b[3][2]={9,8,7,3,2,1}，求其和矩阵 c[3][2]并输出。

10．输入一串字符，计算其中空格的个数。

11．输入一个字符串存入数组中，再将数组内容前后倒置后输出。

12．输入一个字符串存入字符数组中，求出该字符串的长度并输出（不能调用 strlen( )函数）。

13．输入一个字符串存入数组中，再将数组的内容复制到另一个数组中并输出（不能调用 strcpy( )函数）。

14. 输入两个字符串分别存入字符数组中，再将第二个字符串连接到第一个字符串之后并输出（不能调用 strcat( )函数）。

15．从键盘输入一批字符（以@结束）存到数组中，按要求加密并输出。

加密规则：①所有字母均转换为小写；②若是字母 'a'～'y'，则转化为下一个字母；③若是字母 'z'，则转化为 'a'；④其他字符，保持不变。

第 6 章　指　　针

6.1　指针基本概念

6.1.1　访问内存数据

1．直接访问

计算机中的所有数据都是存放在内存（存储器）中的，内存划分为若干存储单元，每个单元可存放一个 8 位二进制数（即 1 字节）。内存中，不同数据类型的数据占用内存的单元数不同，如 char 型变量占 1 个单元，int 型变量占 4 个单元等。

为了正确地访问这些内存存储单元，需要给每个内存存储单元进行编号，这些编号称为地址。内存使用连续编号（地址）为每个单元编址，使每个内存单元对应唯一的地址编号，因此根据一个内存存储单元的编号即可准确地找到该内存单元。

例如，定义三个 int 型变量 a=1、b=2、c=3，编译系统分配给变量 a 的起始地址可能为 0x300000，分配给变量 b 的起始地址可能为 0x300004，分配给变量 c 的起始地址可能为 0x30000C，如图 6-1 所示。

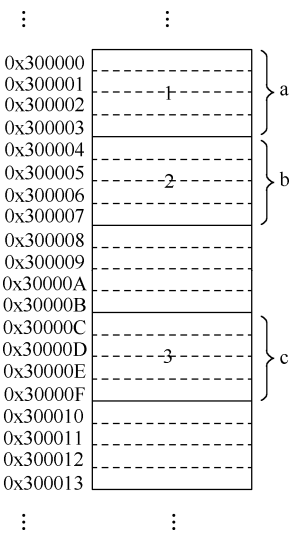


图 6-1　直接访问内存中变量示意图

从图中可以看出，变量 a 占用了 4 字节的内存单元，地址为 0x300000～0x300003，将系统分配给变量的内存单元的起始地址称为变量的地址。例如，变量 a 的地址为 0x300000，变量 b 的地址为 0x300004，变量 c 的地址为 0x30000C。

在程序中对变量进行运算时，如“result=a+b–c”，则通过一张变量名与地址对应关系表（在编译阶段确定），分别找到变量 a 的地址 0x300000，将地址 0x300000～0x300003 中的数据 1 取出，同理，分别找到变量 b 和 c 的地址 0x300004 和 0x30000C，将地址 0x300004～0x300007 和 0x30000C～0x30000F 的数据 2 和 3 取出，然后对数据进行算术运算。这个过程称为变量的“直接访问”。

“直接访问”就是直接使用存放该数据的变量名。使用变量 a 就相当于使用数据 1。

2．间接访问

如果将某一个变量 var 的地址存放在另一个变量 ptr 中，则可以通过变量 ptr 来存取变量 var 的值。

例如，分别用三个变量 pa、pb、pc 来存放变量 a、b、c 的地址，如果要得到变量 a 的值，可以先访问变量 pa，得到变量 a 的地址，再通过该地址找到变量 a 的值，如图 6-2 所示。变量 pa 和变量 a 是通过变量 a 的地址联系在一起的。

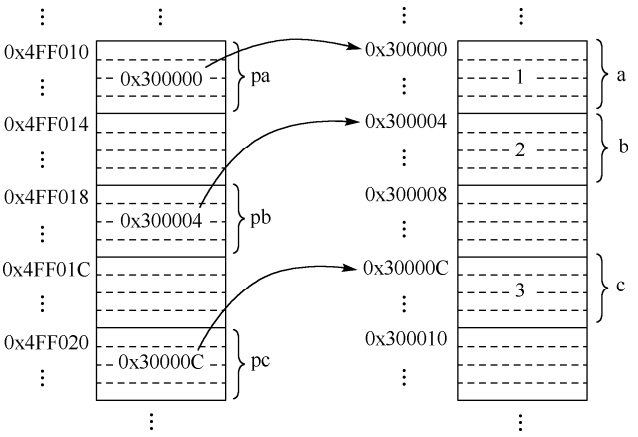


图 6-2 间接访问内存中变量示意图

在进行“result=a+b–c”运算时，首先通过变量 pa 的内容，即 0x300000，找到变量 a 所在的内存位置（即地址 0x300000），将地址 0x300000-0x300003 中的数据 1 取出。同理，分别通过变量 pb 和 pc，找到变量 b 和 c 所在的内存位置（即地址 0x300004 和 0x30000C），将地址 0x300004～0x300007 和 0x30000C～0x30000F 的数据 2 和 3 取出，然后对数据进行算术运算。这个过程称为变量的“间接访问”。

6.1.2 指针定义

通常把内存的地址（编号）称为指针。一个变量的地址称为该变量的指针。例如，地址 0x300000 为变量 a 的指针。

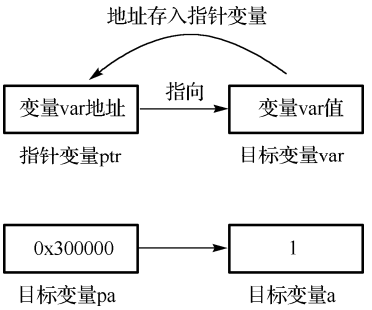


图 6-3 指针变量和目标变量示意图

存放地址（指针）的变量称为指针变量。例如，6.1.1 节中的变量 pa 是一个指针变量。指针是一个变量的地址，指针变量是专门存放变量地址的变量。

一个指针变量的值就是某个目标变量的地址或称为目标变量的指针，如图 6-3 所示。目标变量 var 的地址存放到指针变量 ptr 中，称为指针变量 ptr 指向目标变量 var。例如 6.1.1 节中的指针变量 pa 存放了变量 a 的地址，因此指针变量 pa 指向变量 a。

为了表示指针变量和它所指向的目标变量之间的关系，用“*”符号表示“指向”，例如，pa 是指针变量，*pa 是 pa 指向的变量 a 的内容。因此，语句“a=1;”和“*pa=1;”的作用相同，后者表示将数值 1 赋给指针变量 pa 所指向的变量。

6.2 指 针 变 量

6.2.1 指针变量定义

1. 定义方式

指针变量同普通变量一样，在使用之前必须定义。指针变量定义的一般形式为：“类型说明符 *指针变量名;”，其中“类型说明符”表示本指针变量所指向的目标变量的数据类型，“*”表示这是定

义一个指针变量。如“int *ptr;”表示 ptr 是一个指针变量，它的值是某个 int 型变量的地址，或者说 ptr 指向一个 int 型变量，至于 ptr 指向哪个 int 型变量，应由赋给 ptr 的地址来决定。再如“char *cp;”定义了一个指向字符变量的指针变量 cp。

定义指针变量时应注意如下事项。

（1）可以同时定义多个指针变量，每个指针变量名前都要加上“*”，如“int *p1, *p2;”与“int *p1, p2;”是不同的。前者表示定义了两个指向 int 型变量的指针变量 p1 和 p2，后者表示定义了一个指向 int 型变量的指针变量 p1 和一个 int 型变量 p2。

（2）指针变量名字不带“*”，如“int *p1, *p2;”中指针变量的名字是 p1、p2，不是*p1、*p2。

（3）一个指针变量只能指向定义时所规定类型的变量。如“int *ptr;”不能指向浮点变量，只能指向整型变量。

（4）指针变量定义后，变量的值是随机的，无法确定指向哪个变量，使用该指针变量前必须先赋值或初始化。

2．初始化

指针变量使用之前不仅要定义说明，而且必须赋予具体的值，未经赋值的指针变量不能使用，否则会造成程序崩溃。指针变量初始化的一般形式为：“类型说明符 *指针变量名=初始地址值;”。将初始地址值赋给指针变量，初始地址值为目标变量的地址。

例如，“int a;int *pa=&a;”将变量 a 的地址赋给指针变量 pa，其中的“&”表示对后面的变量做取地址运算。

初始化指针变量时应注意：

（1）取目标变量地址时，目标变量必须已经定义过，且类型必须一致；

（2）虽然地址是一个整型值，但是不能用一个整型值初始化指针变量，如“int *pa=0x300000;”是错误的；

（3）可以使用已初始化的指针变量做初值赋值给另外一个指针变量，如：

```
int count;
int *ptr1=&count;
int *ptr2=ptr1;
```

6.2.2　指针变量引用

1．相关运算符

指针变量有两个运算符：“&”（取地址运算符）和“*”（指针运算符）。

（1）取地址运算符&

取地址运算符“&”将一个变量的地址（指针）赋给一个指针变量，“&”表示取变量的地址，一般形式为：“&变量名;”，如“&a”表示变量 a 的地址。下列语句将变量 i 的地址赋给指针变量 p。

```
int i, *p;
i=3;
p=&i;
```

指针变量的赋值只能赋予地址，决不能赋予任何其他数据，否则将引起错误，如将整数值 0x300000 赋给指针变量“p=0x300000;”是非法的。变量的地址是由编译器分配的，对用户完全透明，用户不知道变量的具体地址。

（2）指针运算符*

指针运算符“*”用来存取指针变量所指向的目标变量的值。如以上代码示例中，“*p”即为 i。

取地址运算符&和指针运算符*均为单目运算符，结合性为自右向左。

2．指针变量的引用

“指针变量名”表示指向目标变量的指针（地址），“*指针变量名”表示指针变量所指向的目标变量的值。不同类型的指针变量可以指向不同类型的变量，但指针变量的值只能是整型值。如下列代码中，pi 指向整型变量 i，修改*pi 的值，即为修改 i 的值。同理，chp 指向字符型变量 ch，pf 指向浮点型变量 f。指针变量 pi、chp 和 pf 的内容均为整型值。

```
int i,*pi=&i;
char ch, *chp=&ch;
float f,*pf=&f;
*pi=99;
*chp='x';
*pf=3.14;
```

把整型变量 i（int i;）的地址赋给指向整型类型的指针变量 pi，有两种方式：①指针变量初始化方式，“int *pi=&I;”；②赋值语句方式，“int *pi; pi=&a;”。

指针变量必须先赋值，再使用，否则会造成程序出现不可预料的结果。

【例 6-1】 程序 6-1：指针变量必须先赋值，再使用。

```
#01: //程序 6-1
#02: #include <stdio.h>
#03: int main(){
#04:     int i=99;
#05:     int *p;
#06:     *p=i;
#07:     printf("*p=%d\n",*p);
#08:
#09:     return 0;
#10: }
```

程序解释：

#06：定义了指针变量 p 之后没有对其赋初值，此时 p 的内容是随机的，当对“*p”赋值时，修改了内存某位置处的值，造成程序运行出错。

程序修改为：

```
#01: //程序 6-1
#02: #include <stdio.h>
#03: int main(){
#04:     int i=99,j;
#05:     int *p;
#06:
#07:     p=&j;
#08:     *p=i;
```

```
#09:    printf("*p=%d,j=%d\n",*p,j);
#10:    return 0;
#11: }
```

程序解释：

#07：对指针变量 p 赋初值，使 p 指向 j。

#08：将 i 的值赋给 p 指向的目标变量。

程序运行结果如下：

```
*p=99,j=99
```

3．运算符&与*关系

运算符“&”用来取目标变量的地址，运算符“*”用来取指针所指向目标变量的内容，两种互为逆运算。当“*&”或“&*”在一起时，具有抵消作用，例如，“*ptr_i=*&I;”相当于“*ptr_i=I;”。如图 6-4 所示，定义 int 型指针变量 ptr_i，并指向 int 型变量 i，则“ptr_i”为指针变量，它的内容是地址量；“*ptr_i”为指针的目标变量，它的内容是数据；“&ptr_i”为指针变量占用内存的地址。

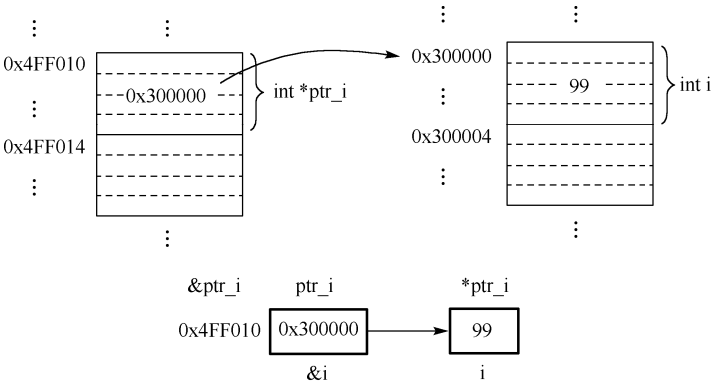


图 6-4　运算符 & 与 * 关系示意图

根据指针变量与其所指向的目标变量之间的关系，等式 “ptr_i = &i = &(*ptr_i);”和“i = *ptr_i = *(&i);”成立，因此语句“i=99;”等价于“*ptr_i=99;”。

4．指针变量的值

与一般的变量相同，指针变量的值也可以改变，即可以改变指针变量的指向。以下的代码使得指针变量 p1 指向变量 a，指针变量 p2 指向变量 b，如图 6-5(a)所示。

```
int a,b;
int *p1,*p2;
a=1;
b=2;
p1=&a;
p2=&b;
```

（1）如果执行语句“p2=p1;”，则将指针变量 p2 的内容修改为指针变量 p1 的内容，即变量 a 的地址，从而指针变量 p2 也指向变量 a，如 6-5图(b)所示。

（2）如果执行语句“*p2=*p1;”，则将指针变量 p2 指向的变量的内容修改为指针变量 p1 指向的变量的内容，即变量 b 的内容修改为变量 a 的内容，如图 6-5(c)所示。

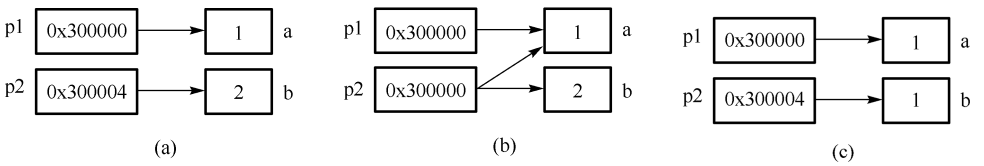


图 6-5 指针变量的值

如同一般变量，在表达式或语句中也可以使用指针变量，例如，如果定义“int a=1,b,*ptr=&a;”则语句“b=2+*ptr;”等价于“b=2+a;”。

另外需要注意，由于运算符“++”和“*”的优先级相同，右结合，因此“b=*ptr++;”等价于“b=*(ptr ++);”，即先对 ptr 的原值进行*运算，得到 a 的值，然后 ptr 的值改变， ptr 不再指向 a 了，可以用两条语句等价：“b=*ptr; ptr++;”。

而“b=(*ptr)++;”等价于“b=a++;”。“b=++*ptr;”等价于“b=++(*ptr);”（即“b=++a”）。

【例 6-2】 程序 6-2：输入两个数，然后从小到大输出。

```
#01: //程序 6-2
#02: #include <stdio.h>
#03: int main(){
#04:     int a,b;
#05:     int *p1=&a,*p2=&b,*p;
#06:
#07:     printf("Input two integers:");
#08:     scanf("%d %d",p1,p2);
#09:
#10:     if(*p1>*p2){
#11:         p=p1;
#12:         p1=p2;
#13:         p2=p;
#14:     }
#15:
#16:     printf("min:%d,max:%d\n",*p1,*p2);
#17:     return 0;
#18: }
```

程序解释：

#05：定义两个指针变量，并分别进行初始化。

#08：将输入的数值存放在指针变量指向的地址（即变量 a 和 b 的地址）。

#10～#14：比较指针变量指向的目标变量的大小（即变量 a 和 b），如果前者大于后者，则交换指针变量的内容。

#16：输出指针变量指向的目标变量的值。

程序运行结果如下：

```
Input two integers:9 2
min:2,max:9
```

6.2.3　空指针和 void 类型指针

1．空指针

空指针也称零指针，即指针变量的值为零，定义为“int *ptr=0;”，即 ptr 指向地址为 0 的单元，系统保证该地址不作它用，表示指针变量的值没有意义。

可以用“NULL”表示空指针，即“ptr=0;”等价于“ptr=NULL;”。空指针常用在条件判断中，例如“while(ptr!=NULL);”。

空指针与未对指针变量赋值不同，未对指针变量 ptr 赋值时，指针变量 ptr 的值是随机的，是不能使用的。而空指针是对指针变量 ptr 赋零值（ptr=NULL），可以使用，只是它不指向具体的变量。

空指针可以避免误操作，防止非法引用指针变量。在指针不使用或指向对象无效时，通常将指针变量置为空指针，否则是“野”指针。使用指针变量时，应先判断它是不是空指针，不是空指针时才有意义，才可以使用它。

2．void 类型指针

如果定义时不指定指针变量是指向哪种类型的数据，而在使用时才能确定，则可以定义为 void 类型的指针，形式为“void *ptr;”。

在使用该指针变量时要进行强制类型转换，将 void 类型指针转换为适当的数据类型。例如：

```
int *p1;
void *p2;
p1=(char *)p2;
```

6.2.4　两重指针

如果一个指针变量中存放的是另一个指针变量的地址，则称这个指针变量为指向指针的指针变量（或者称为两重指针或两级指针）。通过指向指针的指针变量来访问变量，形成“二级间址”。

定义一个两重指针的形式为“类型说明符 **指针变量名;”，指针变量名前有两个“*”，相当于“*(*指针变量名)”，表示该指针变量是指向一个指针型变量的。如“int **pp;”定义一个两重指针 pp，它指向另外一个指针变量 p，而指针变量 p 指向一个 int 型变量，如图 6-6 所示。

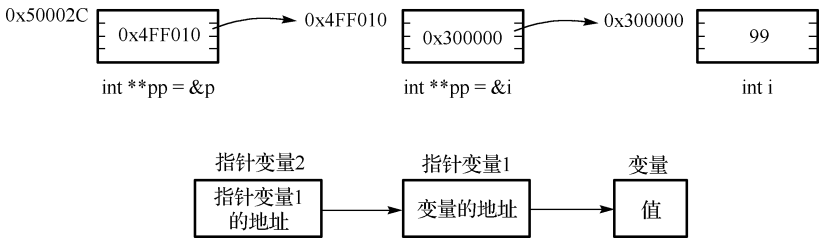


图 6-6　两重指针

图 6-6 用代码表示如下：

```
int i=99;
int *p=&i;
int **pp=&p;
```

对于两重指针 pp，*pp 表示指针变量 p 中的内容，即变量 i 的地址（0x300000），**pp 为变量 i 的内容（99）。因此“*pp”与 p 等价，“**pp”、“*p”和 i 等价。

6.3 指针与数组元素

数组中的元素在内存中被分配一定大小的存储单元，每个数组元素都有一个地址。数组的指针是指整个数组的起始地址，数组元素的指针是数组中对应元素的地址。

6.3.1 指向一维数组元素的指针变量

C 语言为一个数组分配连续的一块内存单元来存储数组元素，数组名表示这块连续内存单元的首地址。数组中的元素按顺序占据这块连续内存单元中的若干，一个数组元素的首地址指它所占据的若干内存单元的起始地址。

定义一个指向数组元素的指针变量的方法与前面相同，只需将数组元素的首地址赋值给它。例如：

```
int array[10];
int *ptr;
ptr=&array[0];
//ptr=array;
//int *ptr=&array[0];
//int *ptr=array;
```

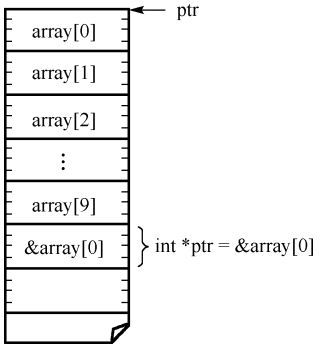


图 6-7 指向一维数组元素的指针变量

在定义了一个指向整型变量的指针变量 ptr 后，使用赋值语句“ptr=&array[0];”把 array[0]元素的地址赋给指针变量 ptr，即 ptr 指向 array 数组的第 0 个元素，如图 6-7 所示。

由于数组名代表数组的首地址，即第 0 个元素的地址，因此语句“ptr=&array[0];”等价于语句“ptr=array;”。也可以用初始化方法对指针变量 ptr 赋值，即“int *ptr=&array[0];”或“int *ptr=array;”。

因此，ptr、array、&array[0]均指向同一个内存单元，都是数组 array 第 0 个元素的地址（可以简称为数组首地址）。

需要注意的是，与指向数组首地址的指针变量 ptr 不同，数组名 array 不是变量，是表示数组首地址的地址常量。

同理，赋值语句“ptr=&array[i];”表示将数组第 i 个元素的地址赋值给指针变量 ptr。

6.3.2 指针变量运算

指针变量是一种特殊的变量，可以参与的运算有三种：赋值运算、加减算术运算、两个指针之间的运算。

1．赋值运算

前面已介绍过指针变量的赋值运算，总结如下：

（1）初始化赋值，如“int *ptr=&a;”（a 要先定义）；

（2）把一个变量地址赋值给指针变量，如“ptr=&a;”（ptr 和 a 要先定义）；

（3）把一个指针变量的值赋值给另一个同类型的指针变量，如“ptr2=ptr1;”；

（4）把数组第 i 个元素的地址赋值给指针变量，如“ptr=&array[i];”。

注意：既不能把一个整数（常量或变量）赋值给指针变量，也不能把指针变量的值赋值给整型变量。指针变量其他类型的赋值运算在后面章节再讲解。

2．加减算术运算

只有在指针变量指向数组元素时，对指针变量进行加减算术运算才有意义。对于指向数组元素的指针变量 ptr，可以对指针变量加上或减去一个整数值 i，即 ptr ± i，以下运算都是合法的：

ptr+i、ptr–i、ptr++、++ptr、ptr– –、– –ptr、ptr+=i、ptr–=i。

指针变量 ptr 加上或减去一个整数 i 的意义，是把指针指向的当前位置（指向数组中某个元素）向前或向后移动 i 个元素位置。因为不同种类型的数组元素所占的字节长度不同，所以指针变量向前、向后移动一个位置和地址值加 1 或减 1 是不同的。指针变量加 1，即向后移动 1 个位置，表示指针变量指向下一个数组元素的首地址，而不是在原地址值的基础上加 1，例如：

```
int array[10];
int *ptr=array;
ptr++;
ptr+=2;
```

指针变量 ptr 初始化时指向数组第 0 个元素的地址，进行“ptr++ ;”运算后，ptr 加 1，指向数组第 1 个元素（array[1]）的地址，进行“ptr+=2;”运算后，ptr 又加 2，即向后移动 2 个位置，此时指向数组第 3 个元素（array[3]）的地址。

3．两个指针之间的运算

只有两个指针变量都指向同一个数组中的元素时，它们之间才能进行运算，两个指针变量可以进行减法运算或关系运算。

两个指针变量不能进行加法运算，两个指针变量减法运算的意义是求两个指针所指向的数组元素之间的元素个数（含第一个指针指向的元素），即两个指针值（地址）相减后再除以数组元素的长度。如下列代码中，进行“n=ptr2–ptr1;”运算后 n 的值为 4。

```
int n, array[10];
int *ptr1=&array[1];
int *ptr2=&array[5];
n=ptr2-ptr1;
```

两个指针变量进行关系运算的意义是比较两个指针所指向的数组元素之间的地址位置关系。如“ptr1==ptr2”比较两个指针变量是否指向同一个数组元素，“ptr1>ptr2”比较 ptr1 指向的数组元素的地址是否大于 ptr2 指向的数组元素的地址（即 ptr1 指向的数组元素是否在 ptr2 指向的数组元素的后面）。以上代码中“ptr1<ptr2”为真，因为 ptr1 指向的数组元素在 ptr2 指向的数组元素的前面。

指针变量也可以与空指针比较，如“ptr==NULL”判断 ptr 是否为空指针，不指向任何变量。

【例 6-3】 程序 6-3：指针变量运算。

```
#01: //程序 6-3
#02: #include <stdio.h>
#03: int main(){
#04:     int a,b,array[5]={9,8,7,6,5};
#05:     int *ptr1=&a,*ptr2=&b;
#06:
```

```
#07:    printf("Input two integers:");
#08:    scanf("%d %d",ptr1,ptr2);
#09:    printf("%d+%d=%d.\n",*ptr1,*ptr2,*ptr1+*ptr2);
#10:    printf("%d*%d=%d.\n",*ptr1,*ptr2,*ptr1**ptr2);
#11:
#12:    ptr1=ptr2=&array[0];
#13:    printf("ptr1==ptr2?%d.\n",ptr1==ptr2);
#14:    ptr1++;
#15:    ptr2+=3;
#16:    printf("*ptr1=%d,*ptr2=%d.\n",*ptr1,*ptr2);
#17:    printf("ptr1>ptr2?%d.\n",ptr1>ptr2);
#18:    printf("ptr1=%p,ptr2=%p.\n",ptr1,ptr2);
#19:    printf("%d elements between ptr2 and ptr1.\n",ptr2-ptr1);
#20:
#21:    return 0;
#22: }
```

程序解释：

#08：scanf( )函数中使用 ptr1 和 ptr2 作为输入数值存放的地址（即变量 a 和 b 的地址）。

#09：使用指针变量对目标变量做加法运算。

#10：使用指针变量对目标变量做乘法运算。“*ptr1**ptr2”等价于“(*ptr1)*(*ptr2)”，即“a*b”。

#13：比较两个指针变量是否指向数组中的同一个元素（为真“1”）。

#14, #15：分别对指针变量 ptr1 和 ptr2 加 1 和 3。

#16：输出当前指针变量指向的元素的值，确认 ptr1 指向数组第 1 个元素，ptr2 指向数组第 3 个元素。

#17：比较 ptr1 指向的元素是否在 ptr2 指向的元素的后面（为假“0”）。

#18：输出两个指针变量的内容，可以看出 ptr2>ptr1。

#19：输出两个指针变量之间的元素个数（为 2），通过指针变量地址值计算“(0028FEF8～0028FEF0)/4=2”也可以验证。

程序运行结果如下：

```
Input two integers:3 4
3+4=7.
3*4=12.
ptr1==ptr2?1.
*ptr1=8,*ptr2=6.
ptr1>ptr2?0.
ptr1=0028FEF0,ptr2=0028FEF8.
2 elements between ptr2 and ptr1.
```

6.3.3 数组元素的表示方法

定义一个 10 个元素的数组 array 和一个指针变量 ptr，并初始化指向数组首地址：

```
int array[10];
int *ptr=&array[0];
```

“ptr+i”或“array+i”就是数组元素 array[i]的地址，或者说它们指向数组 array 的第 i 个元素。

“*(ptr+i)”或“*(array+i)”就是“ptr+i”或“array+i”所指向的数组元素，即 array[i]，因此“array[i]”与“*(array+i)”等价。

指向数组元素的指针变量也可以带下标，如“ptr[i]”与“*(ptr+i)”等价。

引入指针变量后，有两种方法表示数组元素，如图 6-8 所示。

（1）下标法

在介绍数组时，使用下标方法表示数组元素，即“array[i]”的形式，也可以用指针变量表示，即“ptr[i]”表示数组第 i 个元素。

（2）指针法

使用“*(ptr+i)”或“*(array+i)”形式访问数组的第 i 个元素。

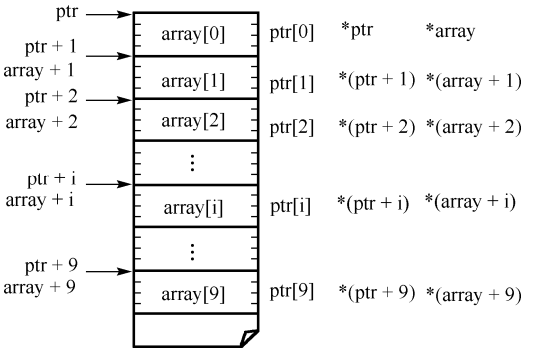


图 6-8　数组元素的表示方法

如果引用数组元素的地址，“ptr+i”与“array+i”等价。如果引用数组元素的内容，“array[i]”、“ptr[i]”、“*(array+i)”和“*(ptr+i)”是等价的。

【例 6-4】 程序 6-4：数组元素的引用方法。

```
#01: //程序 6-4
#02: #include <stdio.h>
#03: #define LEN 4
#04: int main(){
#05:     int array[LEN],*ptr,i;
#06:
#07:     for(i=0;i<LEN;i++)
#08:         array[i]=i*i;
#09:
#10:     ptr=array;
#11:
#12:     for(i=0;i<LEN;i++)
#13:         printf("*(ptr+%d):%d\t",i,*(ptr+i));
#14:     printf("\n");
#15:
#16:     for(i=0;i<LEN;i++)
#17:         printf("*(array+%d):%d\t",i,*(array+i));
#18:     printf("\n");
#19:
#20:     for(i=0;i<LEN;i++)
```

```
#21:        printf("ptr[%d]:%d\t",i,ptr[i]);
#22:    printf("\n");
#23:
#24:    for(i=0;i<LEN;i++)
#25:        printf("array[%d]:%d\t",i,array[i]);
#26:    printf("\n");
#27:
#28:    return 0;
#29: }
```

程序解释：

#07, #08：对数组元素赋值。

#12, #13：使用“*(ptr+i)”形式引用数组元素。

#16, #17：使用“*(array+i)”形式引用数组元素。

#20, #21：使用“ptr[i]”形式引用数组元素。

#24, #25：使用“array [i]”形式引用数组元素。

程序运行结果如下：

```
*(ptr+0):0      *(ptr+1):1      *(ptr+2):4      *(ptr+3):9
*(array+0):0    *(array+1):1    *(array+2):4    *(array+3):9
ptr[0]:0        ptr[1]:1        ptr[2]:4        ptr[3]:9
array[0]:0      array[1]:1      array[2]:4      array[3]:9
```

由于指针变量的值可以改变，指针变量可以指向数组后面的内存单元，使用指针变量时应注意指针变量的当前值，防止对数组后面的内存单元进行操作。

【例 6-5】 程序 6-5：注意指针变量的当前值。

```
#01: //程序 6-5
#02: #include <stdio.h>
#03: #define LEN 5
#04: int main(){
#05:    int array[LEN],*ptr,i;
#06:
#07:    printf("Input %d integers:",LEN);
#08:    ptr=array;
#09:    for(i=0;i<LEN;i++)
#10:        scanf("%d",ptr++);;
#11:    printf("\n");
#12:    ptr=array;
#13:    for(i=0;i<LEN;i++)
#14:       printf("%d\t",*ptr++);
#15:    printf("\n");
#16:    return 0;
#17: }
```

程序解释：

#08：指针变量首先指向数组首地址。

#09, #10：通过指针变量提供的地址，输入数组元素。输入过程中 ptr 的值会发生改变，循环结束后，ptr 指向 array 数组后面的一个内存单元。

#12：修改指针变量的值，再次指向数组首地址。

#13, #14：通过指针变量提供的地址，输出数组元素。输出过程中 ptr 的值会发生改变，循环结束后，ptr 指向 array 数组后面的一个内存单元。

程序运行结果如下：

```
Input 5 integers:9 8 7 6 5

9      8      7      6      5
```

6.3.4　指向二维数组元素的指针变量

1．二维数组元素的地址

本节以二维数组 a[3][4]为例，说明二维数组元素的地址与表示方法。定义二维数组“int a[3][4]={{1,2,3,4},{2,3,4,5},{3,4,5,6}};”，并假定其起始地址为 0x300000，则各元素的首地址如图 6-9 所示。

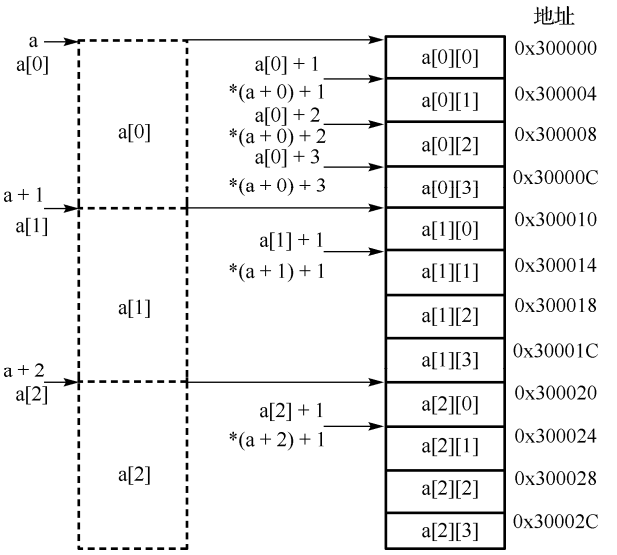


图 6-9　二维数组元素的地址

二维数组是数组的数组，可以把一个二维数组分解为多个一维数组来处理，如数组 a[3][4]可分解为三个一维数组 a[0]、a[1]、a[2]。每个一维数组又含有 4 个元素，如 a[0]数组含有 a[0][0]、a[0][1]、a[0][2]、a[0][3]等 4 个元素。

a 为数组名，是二维数组首元素的地址，这里的首元素不是一个整型变量，而是由 4 个整型元素所组成的一维数组 a[0]，即二维数组第 0 行的首地址（0x300000），同理，a+1 代表二维数组第 1 个元素 a[1]的地址，即二维数组第 1 行的首地址（0x300010）；a+2 代表二维数组第 2 个元素 a[2]的地址，即二维数组第 2 行的首地址（0x300020）。

&a[0]表示第 0 个一维数组的首地址，即第 0 行的首地址，值为 0x300000，等价于 a。同理，&a[1]是第 1 个一维数组的首地址，即第 1 行的首地址，值为 0x300010，等价于 a+1；&a[2]是第 2 个一维数组的首地址，即第 2 行的首地址，值为 0x300020，等价于 a+2。

(a+0)+0 或(a+0)或*a 表示一维数组 a[0]的第 0 个元素的首地址，值虽然也为 0x300000，但是与 a 或 a[0]的意义不同。同理，*(a+0)+1 表示一维数组 a[0]的第 1 个元素的首地址（0x300004）；*(a+0)+2

表示一维数组 a[0]的第 2 个元素的首地址（0x300008）。

a[0]是第 0 个一维数组的数组名，a[0]或 a[0]+0 也表示一维数组 a[0]的第 0 个元素的首地址，值为 0x300000，与“*(a+0)”等价。同理，a[0]+1 表示一维数组 a[0]的第 1 个元素的首地址（0x300004），与“*(a+0)+1”等价；a[0]+2 表示一维数组 a[0]的第 2 个元素的首地址（0x300008），与“*(a+0)+2”等价。

a[1]是第 1 个一维数组的数组名，a[1]或 a[1]+0 表示一维数组 a[1]的第 0 个元素的首地址，值为 0x300010，与“*(a+1)”等价。同理，a[2]是第 2 个一维数组的数组名，a[2]或 a[2]+0 表示一维数组 a[2]的第 0 个元素的首地址，值为 0x300020，与“*(a+2)”等价。

&a[0][0]是二维数组 a 的第 0 行第 0 列元素的首地址，值也为 0x300000。同理，&a[0][1]是二维数组 a 的第 0 行第 1 列元素的首地址（0x300004），&a[1][0]是二维数组 a 的第 1 行第 0 列元素的首地址（0x300010）。

由此可知，在二维数组中，a+i 与&a[i]等价，都表示二维数组分解为一维数组时的第 i 个元素 a[i]的地址，即二维数组第 i 行的首地址，指向二维数组的行。

a[i]与*(a+i) 等价，表示二维数组分解成的一维数组 a[i]的第 0 个元素的首地址。

a[i]+j 与*(a+i)+j 等价，都表示二维数组分解成的一维数组 a[i]的第 j 个元素的首地址，指向二维数组的列。

&a[i][j]表示二维数组 a 的第 i 行第 j 列元素的首地址。

2．二维数组元素的表示

对于指向二维数组 a 的第 i 行第 j 列元素的首地址，加上“*”运算符即可取出二维数组的元素。不同的地址表示方法，二维数组元素有不同的表示形式。

（1）如果用&a[i][j]表示二维数组 a 的第 i 行第 j 列元素的首地址，则二维数组元素表示为：*(&a[i][j])，即 a[i][j]。

（2）如果用 a[i]+j 表示二维数组 a 的第 i 行第 j 列元素的首地址，则二维数组元素表示为：*(a[i]+j)。

（3）如果用*(a+i)+j 表示二维数组 a 的第 i 行第 j 列元素的首地址，则二维数组元素表示为：*(*(a+i)+j)。

（4）如果用相对于 a[0][0]的地址来表示，即&a[0][0]+i*sizeof(类型)+j，则二维数组元素表示为：*(&a[0][0]+i*sizeof(类型)+j)。其中“sizeof(类型)”表示二维数组每个元素的大小。

【例 6-6】 程序 6-6：指向二维数组元素的指针变量。

```
#01: //程序 6-6
#02: #include <stdio.h>
#03: #define ROW 3
#04: #define COL 4
#05: int main(){
#06:     int array[ROW][COL]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
#07:     int *ptr;
#08:
#09:     ptr=*array;
#10:     //ptr=&array[0][0];
#11:     //ptr=array[0];
#12:     for(;ptr<&array[0][0]+ROW*COL;ptr++){
```

```
#13:        if ((ptr-array[0])%COL==0)
#14:            printf("\n");
#15:        printf("%d\t",*ptr);
#16:    }
#17:    return 0;
#18: }
```

程序解释：

#09：将 ptr 赋值为“*(a+i)+j”（i=j=0）的形式，指向二维数组中第 0 行第 0 列元素。

#10：将 ptr 赋值为“&a[i][j]”（i=j=0）的形式，指向二维数组中第 0 行第 0 列元素。

#11：将 ptr 赋值为“a[i]+j”（i=j=0）的形式，指向二维数组中第 0 行第 0 列元素。

以上三行的作用相同，因此只需其中一个即可。

#12：“&array[0][0]+ROW*COL”为相对于 a[0][0]地址的表示形式，指向二维数组第 2 行第 3 列元素（最后一个元素）。

#13, #14：输出二维数组一行元素后换行。

程序运行结果如下：

```
1       3       5       7
9       11      13      15
17      19      21      23
```

6.4　数组指针与指针数组

6.4.1　数组指针

数组指针，也称为“行指针”，是一个指向由若干元素组成的一维数组的指针变量，将该一维数组作为一个整体。数组指针的一般形式为：“类型说明符（*指针变量名）[长度]”，其中“类型说明符”为所指向的一维数组的数据类型，“长度”表示一维数组的长度。“（*指针变量名）”两边需要加括号“( )”且不可省略。

例如，“int(*ptr)[5];”，表示定义了一个数组指针 ptr，指向一个 int 型的一维数组，这个一维数组的长度是 5 个数据元素（用字节表示是：5 * sizeof(int)）。一维数组长度的意义是数组指针的“步长”，即当执行“ptr+1”时，指针 ptr 要跨过 5 个 int 型数据的长度。

数组指针经常用在二维数组中。二维数组是数组的数组，一个二维数组可以分解为多个一维数组，如数组 array[3][4]可分解为三个一维数组 array[0]、array[1]、array[2]，此时可以使用数组指针指向这些一维数组，即二维数组的一行。例如：

```
int array[3][4];
int (*ptr)[4];
ptr=array;
//ptr=&array[0];
ptr++;
```

上述语句定义了一个二维数组 array[3][4]和一个指向含 4 个元素的一维数组的数组指针 ptr；“ptr=array;”或“ptr=&array[0];”将二维数组的第 0 行的地址赋给 ptr；执行“ptr++;”语句后，ptr 指向了二维数组的第 1 行。

使用数组指针时，应注意定义数组指针时给出的一维数组的长度值要与二维数组的列数一致，才能够用于指向二维数组的行。

【例 6-7】 程序 6-7：数组指针示例。

```
#01: //程序 6-7
#02: #include <stdio.h>
#03: #define ROW 3
#04: #define COL 4
#05: int main(){
#06:     int array[ROW][COL]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
#07:     int (*ptr)[COL];
#08:     int i,j;
#09:
#10:     ptr=array;
#11:     //ptr=&array[0];
#12:     for(i=0;i<ROW;i++){
#13:         for(j=0;j<COL;j++)
#14:             printf("%d\t",*(*(ptr+i)+j));
#15:         printf("\n");
#16:     }
#17:     return 0;
#18: }
```

程序解释：

#07：定义一个数组指针 ptr，注意其指向的一维数组长度应等于二维数组的列数。

#10, #11：数组指针 ptr 指向二维数组的第 0 行。两种方法等价。

#12～#16：输出二维数组元素。ptr+i 指向第 i 行，*(ptr+i)指向第 i 行第 0 列元素，*(ptr+i)+j 指向第 i 行第 j 列元素地址。

程序运行结果如下：

```
1       3       5       7
9       11      13      15
17      19      21      23
```

6.4.2 指针数组

一个数组的元素为指针变量，则称为指针数组。指针数组是一组有序的指针变量的集合，其所有元素都是具有相同存储类型和指向相同数据类型目标变量的指针变量。

指针数组的一般形式为：“类型说明符 *数组名[数组长度]”，其中“类型说明符”为指针变量所指向的目标变量类型。如“int *parray[5];”表示 parray 是一个指针数组，它有 5 个数组元素，每个元素都是一个指针变量 parray[i]，指向一个 int 型的目标变量。“parray[0]”或“*parray”表示指针数组第 0 个元素。

可以用一个指针数组来指向一个二维数组，其中指针数组的每个元素指向二维数组每一行第 0 列元素的地址，因此指针数组的大小与二维数组的行数必须一致。例如：

```
int array[3][4];
```

```
int *parray[3];
for(i=0;i<3;i++)
parray[i]=array[i];
```

上述语句定义了一个二维数组 array[3][4]和一个大小为三个元素的指针数组 parray，这三个元素都是指针变量：parray[0]、parray[1]、parray[2]，通过 for 循环分别指向二维数组的对应行的第 0 列元素。如果要访问二维数组中第 i 行第 j 列的元素，“parray[i][j]”、“*(parray[i]+j)”、“*(*(parray+i)+j)”和“(*(parray+i))[j]”都是等价的。

应注意指针数组和数组指针的区别。数组指针是一个指针变量，它在内存中占一个指针的存储空间，定义时“(*指针变量名)”两边的括号不可缺少。而指针数组是一个数组，每个数组元素都是一个指针变量，在内存中以数组的形式占用多个指针的存储空间。

例如，“int(*p)[3];”表示一个指向长度为 3 的一维数组的指针变量 p；“int*p[3];”表示 p 是一个指针数组，三个元素 p[0]、p[1]、p[2]均为指针变量。

【例 6-8】 程序 6-8：指针数组示例。

```
#01: //程序 6-8
#02: #include <stdio.h>
#03: #define ROW 3
#04: #define COL 4
#05: int main(){
#06:     int array[ROW][COL]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
#07:     int *parray[ROW];
#08:     int i,j;
#09:
#10:     for(i=0;i<ROW;i++){
#11:         parray[i]=array[i];
#12:         //parray[i]=*(array+i);
#13:         for(j=0;j<COL;j++)
#14:             printf("%d\t",*(parray[i]+j));
#15:         printf("\n");
#16:     }
#17:     return 0;
#18: }
```

程序解释：

#07：定义一个指针数组 parray，注意数组长度应等于二维数组的行数。

#11, #12：将二维数组的第 i 行第 0 列元素的地址赋值给指针数组 parray 的第 i 个元素。两种方法等价。

#13, #14：输出二维数组元素。parray[i] 指向第 i 行第 0 列元素，parray[i]+j 指向第 i 行第 j 列元素。

程序运行结果如下：

```
1       3       5       7
9       11      13      15
17      19      21      23
```

6.5 指针与字符串

可以用两种方式表示一个字符串：①字符数组；②字符指针。

（1）字符数组

前面已经介绍过，字符数组可以用来表示字符串，如“char string[]="Hello world! ";”。下列代码将输出字符串“Hello world!”和“world!”。

```
char string[]="Hello world!";
printf("%s\n", string);
printf("%s\n", string+6);
```

（2）字符指针

字符指针即指向字符变量的指针变量，可以将一个字符变量的地址赋值给字符指针，也可以将一个字符串的首地址赋值给字符指针。如“char ch,*ptr=&ch;”表示字符指针 ptr 是一个指向字符变量 ch 的指针变量；“char *ptr=" Hello world! ";”则表示字符指针 ptr 是一个指向字符串常量"Hello world! "的指针变量，即字符串的首地址赋值给字符指针 ptr。下列代码将输出字符串" Hello world! "和" world! "。

```
char* string="Hello world!";
printf("%s\n", string);
printf("%s\n", string+6);
```

“char *string=" Hello world! ";”表示 string 是一个字符指针变量，初始化时把字符串的首地址赋给 string，等价于两条语句：“char *string;”和“string=" Hello world!";”。“string+6”将字符指针变量 string 向后移动 6 个字符的位置，此时指向字符“w”的地址。

注意语句“string="Hello world!";”并不是把整个字符串装入指针变量，而是把存放该字符串的内存单元的首地址赋值给指针变量。

字符数组（“char str[20];”）与字符指针（“char *cp;”）有如下区别。

（1）定义的意义不同。str 表示数组由若干元素组成，每个元素放一个字符；cp 表示指向一个字符串或字符的地址，是一个指针变量，存放的是字符串首地址。

（2）只能将一个字符串初始化给一个字符数组，不能将一个字符串赋值给一个字符数组。如“str="Hello world!";”是错误的，不能在赋值语句中整体赋值；可以用字符指针来完成，语句“cp=" Hello world! ";”是正确的。

（3）str 是一个地址常量；cp 是一个地址变量。

（4）用在 scanf( )函数中接收输入的字符串时，“scanf(" %s ",str);”是正确的，“scanf(" %s", cp);”是错误的，必须先使它指向确定的字符数组，才能接收输入的字符串，可修改为：

```
char *cp,str[10];
cp=str;
scanf(“%s”,cp);
```

【例 6-9】 程序 6-9：字符串指针示例：检测输入的字符串中是否有空格。

```
#01: //程序 6-9
#02: #include <stdio.h>
#03: #define LEN 80
#04: int main(){
```

```
#05:    char string[LEN],*pstr;
#06:
#07:    pstr=string;
#08:    printf("Input a string:");
#09:    gets(pstr);
#10:
#11:    while(*pstr++!='\0')
#12:       if (*pstr==' '){
#13:          printf("Find a space in '%s'.\n",string);
#14:          break;
#15:       }
#16:
#17:    if (*pstr=='\0')
#18:       printf("No space in '%s'.\n",string);
#19:    return 0;
#20: }
```

程序解释：

#07：字符指针 pstr 指向字符数组 string 的首地址，以接收输入的字符串。

#09：使用 gets( )函数接收字符串，gets( )函数需要一个地址参数。

#11：判断 pstr 是否指向字符串的结束位置（\0），并将指针加 1（移向字符串下一个字符位置）。

#12～#15：通过*pstr 读取 pstr 处的字符，判断是否是空格字符，如果是空格，则结束循环。

#17, #18：如果字符串中没有空格字符，则 while 循环后 pstr 将指向字符串的结束位置（\0），“*pstr== '\0'”为真。

程序运行结果如下：

```
Input a string:hello world
Find a space in 'hello world'.
```

如果要同时表示多个字符串，可以使用指针数组。指针数组的每个元素指向一个字符串的首地址。指向多个字符串的指针数组的初始化很简单，如下列代码初始化一个指针数组 company 后，可使用 for 循环依次输出每个字符串。

```
char *company[]={"Miscrosoft",
     "Apple",
     "Google",
     "Facebook",
     "Amazon"};
int i;
for(i=0;i<5;i++)
  printf("%s\n",company[i]);
```

6.6　程 序 示 例

【例 6-10】 程序 6-10：输入 *N* 个整数，一边输入一边按从小到大的顺序进行排序，即输入每个整数时，插入到已排好序的数组中。

```
#01: //程序 6-10
#02: #include <stdio.h>
#03: #define N 5
#04: int main(){
#05:     int i,j,k,num,a[N]={0};
#06:     int *ptr=a,*p;
#07:
#08:     for(i=0;i<N;i++){
#09:        printf("Input a number:");
#10:        scanf("%d",&num);
#11:
#12:        for(j=0;j<i;j++)
#13:           if(num<*(ptr+j)){
#14:              p=ptr+i;
#15:              for(k=i;k>j;k--){
#16:                 *p=*(p-1);
#17:                 p--;
#18:              }
#19:              break;
#20:           }
#21:
#22:        *(ptr+j)=num;
#23:
#24:        printf("Now the numbers in array:\n");
#25:        for(j=0; j<i+1; j++)
#26:           printf("%d\t",*(ptr+j));
#27:         printf("\n");
#28:     }
#29:     return 0;
#30: }
```

程序解释：

#06：指针变量 ptr 指向数组 a 的第 0 个元素地址。

#12～#20：依次输入 *N* 个整数，将输入的整数插入到已经有序的有 i 个元素的数组中。

#13：从第 0 个元素开始顺序比较当前输入整数 num 与数组第 j 个元素的大小。如果输入整数小于数组第 j 个元素（*(ptr+j)），则将输入整数插入到当前位置（数组的第 j 个元素）。

#14：指针变量 p 指向数组最后一个元素的后面一个位置（例如，如果当前数组有 i 个元素，则 p 指向 a[i]）。

#15～#17：将数组下标从 j 到 i–1 的元素移动到数组下标 j+1 到 i 的位置（空出数组下标为 j 的位置）。

#22：将当前输入整数 num 插入到数组第 j 个元素位置（ptr+j）。

#25, #26：将当前数组的元素（已经有序）全部输出。

程序运行结果如下：

```
Input a number:6
Now the numbers in array:
```

```
6
Input a number:3
Now the numbers in array:
3      6
Input a number:9
Now the numbers in array:
3      6      9
Input a number:8
Now the numbers in array:
3      6      8      9
Input a number:7
Now the numbers in array:
3      6      7      8      9
```

【例 6-11】 程序 6-11：输入一个字符串，判断是否为回文（palindrome）。

```
#01: //程序 6-11
#02: #include <stdio.h>
#03: #include <string.h>
#04: #define LEN 80
#05: int main(){
#06:    char string[LEN];
#07:    int flag = 1;
#08:    char *start,*end;
#09:
#10:    start=string;
#11:    printf("Input a string:");
#12:    gets(start);
#13:    end=string+strlen(string)-1;
#14:
#15:    for(;start <= end; start++, end--)
#16:       if(*start != *end){
#17:          flag = 0;
#18:          break;
#19:       }
#20:
#21:    if (flag==1)
#22:      printf("'%s' is a palindrome string.\n",string);
#23:    else
#24:      printf("'%s' is NOT a palindrome string.\n",string);
#25:    return 0;
#26: }
```

程序解释：

#07：定义一个 int 型变量 flag，用来指示 string 是否为一个回文。

#10：字符指针 start 指向字符串中第一个字符。

#13：字符指针 end 指向字符串中最后一个字符（不含 '\0'）。

#15：将 start 指针向后移动，end 指针向前移动，当两个指针相遇时结束。

#16～#18：如果 start 位置的字符与 end 位置的字符不相同，则不是一个回文，将 flag 置 0 并退出循环。

#21～#24：根据 flag 的值输出是否为回文。

程序运行结果如下：

```
Input a string:madam
'madam' is a palindrome string.
```

上机实验：指针程序设计应用

本实验掌握 C 语言程序的指针变量的定义与引用，熟练使用指针与数组、指针与字符串等知识编写应用程序。

（1）输入一个字符串，求字符串的长度（不调用 strlen( )函数）。

程序示例：

```
#include <stdio.h>
#define LEN 80
int main(){
    char string[LEN],*ptr;
    int num=0;

    ptr=string;
    printf("Input a string:");
    gets(ptr);

    while(*ptr++!='\0')
        num++;

    printf("The length of '%s' is %d.\n",string,num);
    return 0;
}
```

（2）输入 *N* 个整数，将其按相反的顺序存放。

程序示例：

```
#include<stdio.h>
#define LEN 4
int main() {
    int i,temp,array[LEN];
    int *start,*end;

    printf("Input %d integers:",LEN);
    start=array;
    for(i=0;i<LEN;i++,start++){
        scanf("%d",start);
    }
```

```
    start=array;
    end=array+LEN-1;
    for(;start<end;start++,end--){
        temp=*start;
        *start=*end;
        *end=temp;
    }

    printf("Inverted array:\n");
    start=array;
    while(start<array+LEN)
        printf("%d\t",*start++);
    printf("\n");
    return 0;
}
```

习　　题

1．用指针方法实现：输入两个整数，输出其中较大的数。

2．用指针方法实现：输入三个整数，将这三个数按从小到大的顺序输出。

3．用指针方法实现：计算一个数组中所有元素的和及平均值。

4．用指针方法实现：输入一个 10 个元素的数组，输出数组的第 3～7 个元素。

5．用指针方法实现：输入一个 10 个元素的数组，输出数组的奇数位置的元素。

6．用指针方法实现：对 10 个整数按从小到大的顺序排序。

7．用指针方法实现：输入一个 10 个元素的数组，输出数组中的最大值和最小值。

8．用指针方法实现：输入一个 10 个元素的数组，将每个元素乘以 2 并输出。

9．用指针方法实现：输入一个二维数组，再输入行号和列号，输出对应的数值。

10．用指针方法实现：输入一个二维数组，输出二维数组中的最大值和最小值。

11．用指针方法实现：有 *N* 个人围成一圈，顺序排号，从第一个人开始报数（1→*M* 报数），凡是报到 *M* 的人退出圈子，则最后留下的人原来排在第几号？

12．用指针方法实现：输入一串字符，计算其中空格的个数。

13．用指针方法实现：输入一个字符串存入数组中，再将数组的内容复制到另一个数组中并输出（不能调用 strcpy( )函数）。

14．用指针方法实现：输入两个字符串分别存入字符数组中，再将第二个字符串连接到第一个字符串之后并输出（不能调用 strcat( )函数）。

15．用指针方法实现：首先用指针数组表示出 12 个月份的英文名称，然后输入一个月份号，输出对应的英文月号。如输入“10”，输出“October”。

16．用指针方法实现：从键盘输入一批字符（以@结束）存到数组中，按要求加密并输出。

加密规则：①所有字母均转换为小写；②若是字母'a'～'y'，则转化为下一个字母；③若是'z'，则转化为 'a'；④其他字符，保持不变。

第7章　函　　数

7.1　函数基本知识

在C语言程序设计时，为了简化程序的设计，缩短开发周期，并使得程序具有易维护、易扩充的功能，可以将一个较大的程序按功能划分成一些较小的模块。每个模块相对独立、结构清晰，降低了程序设计的复杂性，这些较小的模块可以用函数来表示。

C语言的源程序是由函数组成的，通过对不同函数模块的调用实现不同特定的功能。由于C语言程序的功能都是由各种各样的函数来完成的，因此把C语言称为函数式语言。通过函数能够实现结构化程序设计，使程序的层次结构清晰，避免程序开发的重复劳动，便于程序的开发与调试。

对于C语言程序，有且只能有一个名为main的主函数，程序的执行总是从main函数开始，在main中结束的。

7.1.1　函数分类

C语言不仅提供了极为丰富的库函数，还允许用户编写自己定义的函数。从不同的角度可以对函数做不同的分类。

（1）从用户角度，函数可分为标准函数（或库函数）和用户自定义函数两类。

标准函数无须用户定义，由系统提供，放在头文件中，用户只需在程序开始位置添加相应的头文件即可（使用“#include”）。如“printf( )”、“getchar( )”和“strlen( )”等都是标准函数。

用户自定义函数是由用户按实际需要而编写的函数。需要在程序中先定义函数功能，然后才能使用该函数。本章将讲述如何定义和使用函数。

（2）从函数有无返回结果，函数可分为有返回值函数和无返回值函数两类。

有返回值函数在被调用执行后，向调用者返回一个执行结果，称为函数返回值。例如，计算一个数的平方根函数“sqrt( )”将计算结果返回给调用者。用户自定义函数如果需要有返回值，则在函数定义时必须明确返回值的数据类型。

无返回值函数用于完成某项特定的处理任务，执行完成后不向调用者返回任何值。例如，下列函数只在屏幕上输出5个“*”，不返回结果给调用者。

```
void printstar(){
   printf("*****\n");
}
```

用户自定义函数如果不需要返回值，在定义函数时指定它的返回类型为“void”，即“空类型”。

（3）从主调函数和被调用函数之间有无数据传送的角度，函数可分为无参函数和有参函数两类。

无参函数在函数调用时不带参数，即主调函数和被调用函数之间不进行参数传递，通常用来完成指定的功能，如上面定义的函数“printstar( )”。

有参函数在函数调用时必须带参数，即主调函数通过参数向被调用函数传递数据，如“printf( )”函数需要传递输出的格式和值。

标准函数（或库函数）是系统自带的标准库函数，根据不同的功能被放在不同的头文件中，如表 7-1 所示。更多函数说明可查阅附录 B.3 或相关手册。

表 7-1　标 准 函 数

函 数 类 别	头　文　件	说　　明	举　　例
输入/输出函数	stdio.h	用于输入或输出	printf( )、fprintf( )
数学函数	math.h	用于数学函数计算	abs( )、sqrt( )、sin( )
字符函数	ctype.h	用于对字符进行分类和判断	isdigit( )、isupper( )
字符串函数	string.h	用于字符串操作和处理	strcpy( )、strlen( )
转换函数	stdlib.h/ ctype.h	用于字符量和数字量、大小写转换	atoi( )、tolower( )
日期/时间函数	time.h	用于日期/时间转换操作	time( )、ctime( )
内存管理函数	stdlib.h	用于动态存储分配	malloc( )、free( )

7.1.2　函数定义

函数必须先定义，然后才能被调用。函数定义的一般格式为：

```
函数类型标识符 函数名(形式参数列表){
   [局部变量说明部分]
   语句部分
   }
```

其中，“函数类型标识符 函数名(形式参数列表)”称为“函数头”；“{}”中的内容称为“函数体”，“{}”不可省略。

“函数类型标识符”指明了该函数的类型，函数类型是函数返回值的类型。“函数名”由用户自定义，要求是合法的标识符。“函数名”后有一个括号“( )”，不可省略。

“函数体”开始部分是“局部变量说明部分”，用来定义本函数内部需要使用的变量，如果没有用到其他变量，该部分可以省略。“局部变量”是指函数内部定义的变量，其有效范围仅限于所在函数的内部，离开函数体则无意义。剩余部分为函数的语句，用来说明函数的功能。

如果不要求函数有返回值，则“函数类型标识符”使用“void”。如果要求函数有返回值，则在函数体中至少应有一个 return 语句，用于把函数的值返回给主调函数。

如果不要求函数有参数，则“形式参数列表”可以省略，或者使用“void”表示；否则，“形式参数列表”指明该函数的形式参数的类型说明和变量名。在“形式参数列表”中给出的参数称为“形式参数”，它们可以是各种类型的变量，之间用逗号隔开。如“int max(int x, int y);”中，max( ) 函数需要两个参数，分别是一个 int 型的 x 和一个 int 型的 y。注意每个变量都需要类型说明，如果写成“int max(int x, y);”则是错误的。

【例 7-1】 函数定义示例。

```
#01: void hello(){
#02:     printf("Hello,world!\n");
#03: }
#04:
#05: int max(int x,int y){
#06:     int result;
#07:     result =x>y?x:y;
```

```
#08:     return(result);
#09: }
```

程序解释：

#01～#03：定义了一个无参数和无返回值的函数 hello( )，其功能是输出一个字符串“Hello,world!”。

#05～#09：定义了一个有两个 int 型参数的函数 max( )，函数返回值为 int 型，其功能是求两个数中的较大数。

#06：在 max 函数体中定义了一个局部变量 result，其只在 max( )函数中有效。

#08：return 语句是把 result 的值作为函数的值返回给主调函数。

函数定义是独立平行的，在一个函数的函数体内部不能定义另外一个函数。C 语言中，每个函数定义的顺序没有要求，一个函数定义可以放在任意位置（只要不在其他函数体内部即可）。

7.2 函数参数与返回值

7.2.1 形参与实参

函数的参数用于在主调函数和被调用函数之间传递数据。定义函数时，函数名后面括号中的变量名称为“形式参数”（简称“形参”）；调用函数时，函数名后面括号中的表达式称为“实际参数”（简称“实参”）。进行函数调用时，主调函数将把实参的值传送给形参，供被调用函数使用。

函数定义中的“形参”是局部变量，在整个函数体内都可以使用，但是离开该函数则无效，不能使用。主调函数中的“实参”也是局部变量，只有在主调函数中有效，因此在被调函数中不能使用“实参”。形参和实参的作用是进行数据传递，当函数调用时，主调函数把“实参”的值传递给被调用函数的“形参”中，从而实现主调函数向被调用函数的数据传递。

【例 7-2】 程序 7-1：形参与实参实现数据传递。

```
#01: //程序 7-1
#02: #include <stdio.h>
#03: int max(int x,int y){
#04:     int result;
#05:     result =x>y?x:y;
#06:     return(result);
#07: }
#08:
#09: int main() {
#10:     int a,b,c;
#11:
#12:     printf("Input two integers:");
#13:     scanf("%d %d",&a,&b);
#14:
#15:     c=max(a,b);
#16:     printf("Max is:%d.\n",c);
#17:     return 0;
#18: }
```

程序解释：

#03～#07：定义 max 函数，形参为变量 x、y。

#15：main( )函数调用 max( )函数，通过把实参 a、b 的值传递给 max( )函数的形参 x、y，然后 max( )函数将执行结果 result 返回给变量 c。具体过程如图 7-1 所示。

程序运行结果如下：

```
Input two integers:5 9
Max is:9.
```

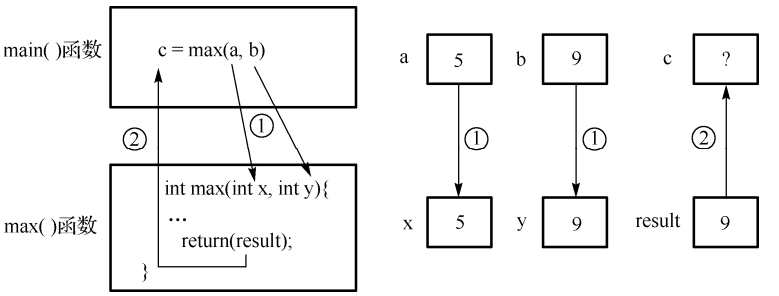


图 7-1　形参与实参实现数据传递

图中，在 main( )函数中，假定输入了两个整数 5、9，则变量 a 的值为 5，变量 b 的值为 9。当执行语句“c=max(a,b);”时，将调用定义的 max( )函数，此时，实参 a 和 b 将自己的值分别传递给对应的形参，即变量 x 和 y，因此 x 值为 5，y 值为 9。

在 max( )函数中，通过执行相应语句，计算出 result 的值为 9。当执行语句“return (result);”时，将变量 result 的值传递给 main( )函数中的变量 c，从而得到 c 的值为 9。

对于“形参”和“实参”应注意：

（1）在定义函数时必须说明形参的数据类型，形参只能是变量、数组、指针或自定义数据类型（第 8 章讲述）等。

（2）形参在函数被调用前不分配内存空间，当函数被调用时，临时为形参分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在被调用函数内部有效，函数调用结束返回主调函数后，就不能再使用该形参。

（3）实参在函数调用时必须有确定的值，以便传递给形参，实参可以是常量、变量、函数、指针或表达式等类型。

（4）形参与实参在数量上必须相同，相对应的类型必须一致。如果形参与实参类型不一致，自动按形参类型进行转换（称为“函数调用转换”）。

（5）尽管实参和形参的名字可以相同，但为了避免混淆，建议实参和对应的形参不要用完全相同的名字。

（6）函数调用时，实参与形参之间的数据传递是单向的，即只能把实参的值传递给形参，而不能把形参的值反向地传递给实参。因此在函数调用过程中，无论形参的值怎么改变，实参的值都不会变化。

【例 7-3】 程序 7-2：实参与形参之间的数据传递是单向的。

```
#01: //程序 7-2
#02: #include <stdio.h>
#03: int func(int x){
#04:     x--;
#05:     printf("x=%d\n",x);
```

```
#06: }
#07:
#08: int main() {
#09:     int num;
#10:
#11:     printf("Input a number:");
#12:     scanf("%d",&num);
#13:
#14:     func(num);
#15:     printf("num=%d\n",num);
#16:     return 0;
#17: }
```

程序解释：

#04：func( )函数将形参 x 的值减 1，其值的改变不会反向传递给 num。

#14：如果输入 num 的值为 5，则调用函数“func(num)”时，实参 num 将 5 传递给形参 x，此时 x 值为 5。

#15：无论 func( )函数怎么改变形参 x 的值，实参 num 的值不会发生改变。

程序运行结果如下：

```
Input a number:5
x=4
num=5
```

7.2.2 函数返回值

函数的返回值，即函数值，是一个确定的值。函数值是指函数被调用之后，执行函数体中的程序代码所得的，并将返回给主调函数的值。函数通过返回语句“return”使程序从被调用函数返回到主调函数中，同时将函数值返回给主调函数，return 的一般形式为：

return (表达式);

或 return 表达式;

其功能是计算表达式的值，并返回给主调函数。return 语句后面的值可以是一个表达式，其括号可以省略。

如果一个函数有返回值，则必须使用 return 语句将函数值返回。一个函数中可以有一个以上的 return 语句，但不论执行到哪个 return 语句，都会结束函数的调用，返回到主调函数中，因此一次函数调用只能返回一个函数值。如以下语句中，如果变量 x 大于变量 y，则将变量 x 的值返回给主调函数，结束函数的调用；否则将变量 y 的值返回给主调函数，结束函数的调用。

```
int max(int x,int y){
  if (x>y)
     return x;
   else
     return y;
   }
```

函数的类型即是函数值的类型，如定义 max( )函数时是 int 型，则说明返回的函数值是 int 型的。

如果没有说明函数的类型，则默认是 int 型。return 语句中表达式的值应与函数类型一致，如果不一致，则自动进行类型转换（仅限数值型数据），将表达式的值转换为函数类型的值。

如果函数中没有使用 return 语句，则函数的返回值是一个不确定的数值。如果一个函数不需要返回值，可以用 void 将函数类型定义为空类型，此时就不能在主调函数中使用被调用函数的函数值。也就意味函数类型为 void 的函数调用不能出现在表达式中，只能作为一条函数语句使用。

【例 7-4】 程序 7-3：函数返回值示例。

```
#01: //程序 7-3
#02: #include <stdio.h>
#03: void outputstar(){
#04:    printf("*****\n");
#05: }
#06: int func(int x){
#07:    printf("x=%d\n",x);
#08: }
#09: int addup(int x,int y){
#10:    return (x+y);
#11: }
#12:
#13: int main() {
#14:    int a=2,b=3,c;
#15:
#16:    outputstar();
#17:    c=func(a);
#18:    printf("c=%d\n",c);
#19:    c=func(b);
#20:    printf("c=%d\n",c);
#21:
#22:    c=addup(a,b);
#23:    printf("c=%d\n",c);
#24:    return 0;
#25: }
```

程序解释：

#03～#05：定义一个函数类型为 void 的函数。

#07～#08：定义一个 int 类型的函数，但是没有使用 return 语句。

#09～#11：定义一个返回值为表达式 (x+y) 的函数。

#16：使用函数语句形式调用 void 类型的函数。

#17～#20：对于没有 return 语句的函数，其返回值不确定（此处为 4）。

#21：调用函数 addup( )，该函数将表达式 (x+y) 的值返回给变量 c。

程序运行结果如下：

```
*****
x=2
c=4
```

```
x=3
c=4
c=5
```

7.3 函 数 调 用

7.3.1 函数调用形式

函数调用的一般形式为：“函数名（实参列表）”，如果函数无参数，则调用时“实参列表”为空；“实参”与“形参”的个数相等，类型一致，并按顺序一一对应；“实参列表”中的参数用逗号分隔，可以是常数、变量、数组、指针或表达式等，参数的求值顺序（自左至右还是自右至左依次求参数的值）因编译器而定。

1．调用形式

函数的调用方式有以下三种。

（1）函数语句。函数调用一般形式加上分号“;”形成函数语句，如“printf(" Hello world!\n ");”、“scanf(" %d ",&num);”等以函数语句方式调用函数。

（2）函数表达式。该方式要求函数有返回值，函数作为表达式一部分，以函数返回值方式参与表达式运算，如“z=max(x,y)*2;”。

（3）函数参数。函数的调用作为一个函数的实参，也要求函数有返回值，函数作为另一个函数调用的实际参数出现，如“printf(" %d\n", strlen(str));”、“m=max(a,max(b,c));”等。

函数调用的执行过程如下：①主调函数计算每个实参表达式的值；②按照对应顺序，将实参的值一一传递给形参；③执行被调用函数；④当遇到 return 语句时，计算 return 后面表达式的值，然后返回主调函数中。

【例 7-5】 程序 7-4：函数调用过程示例。

```
#01: //程序 7-4
#02: #include <stdio.h>
#03: float fabs(float f){
#04:     return (f>0?f:-f);
#05: }
#06:
#07: int main() {
#08:     float x,y;
#09:
#10:     printf("Input a float:");
#11:     scanf("%f",&x);
#12:     y=fabs(x*10);
#13:     printf("x=%.3f,|10x|=%.3f\n",x,y);
#14:     return 0;
#15: }
```

程序解释（按照函数调用过程）：

#12：计算实参表达式“x*10”的值。如果输入–3.14，则实参值为–31.4。

#03：将实参值–31.4 传递给形参 f，此时 f 值为–31.4。

#04：执行函数 fabs，计算表达式“f>0?f: –f”的值，此时为 31.4。

#12：被调用函数通过 return 将函数值 31.4 返回给主调函数，通过赋值语句赋值给变量 y，此时 y 的值为 31.4。

程序运行结果如下：

```
Input a float:-3.14
x=-3.140,|10x|=31.400
```

2．函数声明

函数声明是对函数原型的说明，函数原型提供如下信息：①函数返回值的类型；②函数参数的个数；③每个参数的类型；④参数的顺序。被调用函数必须是已存在的函数，如果是库函数，其函数声明在“.h”头文件中，使用“#include <*.h>”包含声明；对于用户自定义函数，当被调用函数定义的位置在主调函数后面时，需要有函数声明。

函数声明的位置通常在程序的数据说明部分，其一般形式为：“函数类型 函数名(形参类型[形参名],…);”，其中“形参名”可以省略。

【例 7-6】 程序 7-5：求 *x* 的 *n* 次方。

```
#01: //程序 7-5
#02: #include <stdio.h>
#03: double power(double,int);
#04: //double power(double x,int n);
#05: int main() {
#06:     double value,result;
#07:     int num;
#08:
#09:     printf("Input a double:");
#10:     scanf("%lf",&value);
#11:     printf("Input a power(int):");
#12:     scanf("%d",&num);
#13:
#14:     result=power(value,num);
#15:     printf("%.3lf**%d=%.3lf\n",value,num,result);
#16:     return 0;
#17: }
#18:
#19: double power(double x,int n){
#20:     double val = 1.0;
#21:     while (n--)
#22:         val = val*x;
#23:     return (val);
#24: }
```

程序解释：

#03, #04：对 power 函数进行函数声明，两种声明方式等价。

#19～#24：power 函数的定义。

程序运行结果如下：

```
Input a double:2.5
Input a power(int):3
2.500**3=15.625
```

函数定义与函数声明不同，函数定义是一个完整的函数单元，包含函数类型、函数名、形参及形参类型、函数体等部分。函数声明只是对编译器的一个说明，不包含函数体（或形参），是一条语句，必须以分号“;”结尾。

若被调用函数的定义在主调函数之前，可省略对被调函数的函数声明，此外，如果被调用函数返回值是整型或字符型，可以不对被调用函数做函数声明而直接调用，系统自动按整型处理。

7.3.2 函数嵌套调用

C 语言不允许嵌套的函数定义，但可以嵌套调用函数，即在一个函数定义中又调用其他函数，例如，下列代码即为两层嵌套调用。

```
void fun2(){
   …
}
void fun1(){
    fun2();
}
int main(){
    fun1();
}
```

其关系表示如图 7-2 所示，为两层嵌套，图中数字表示其执行过程：在执行 main( )函数时，遇到调用 fun1( )函数的语句时，转去执行 fun1( )函数。在 fun1( )函数中遇到调用 fun2( )函数的语句时，又转去执行 fun2( )函数。fun2( )函数执行结束后，返回到 fun1( )函数中接着以前的位置继续执行，fun1( )函数执行结束后，返回到 main( )函数中接着以前的位置继续执行。

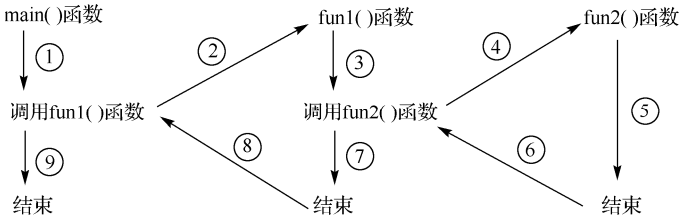


图 7-2 函数嵌套调用关系表示

【例 7-7】 程序 7-6：函数嵌套调用示例，输入两个整数 x、y，求 x^2+y^3 的值。

```
#01: //程序 7-6
#02: #include <stdio.h>
#03: int summary(int,int);
#04: int square(int);
#05: int cube(int);
#06: int main() {
#07:     int x,y;
#08:
```

```
#09:    printf("Input two integers:");
#10:    scanf("%d %d",&x,&y);
#11:    printf("square(%d)+cube(%d)=%d\n",x,y,summary(x,y));
#12:
#13:    return 0;
#14: }
#15:
#16: int summary(int x,int y){
#17:    return (square(x)+cube(y));
#18: }
#19: int square(int m){
#20:    return (m*m);
#21: }
#22: int cube(int m){
#23:    return (m*m*m);
#24: }
```

程序解释：

#03～#05：函数声明部分。

#11：main( )函数调用 summary( )函数。

#16～#18：summary( )函数定义，其中调用了 square( )函数和 cube( )函数。

#19～#21：square( )函数定义，返回平方值。

#22～#24：cube( )函数定义，返回立方值。

函数调用关系如图 7-3 所示。

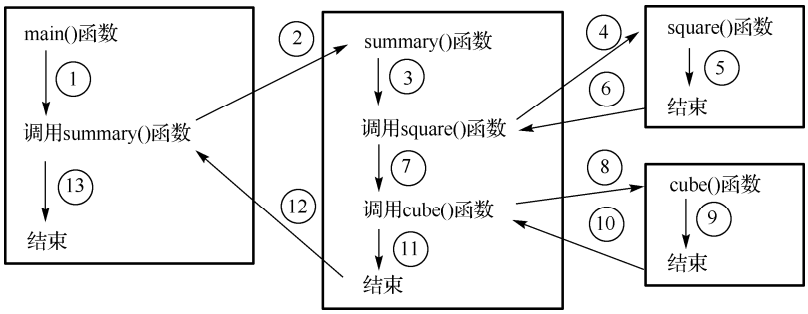


图 7-3　函数嵌套调用关系

程序运行结果如下：

```
Input two integers:10 20
square(10)+cube(20)=8100
```

7.3.3　函数递归调用

一个函数直接或间接地调用其自身，叫函数的递归调用，该函数称为递归函数，例如，下列代码在 fun( )函数中又直接调用了 fun( )函数。

```
int fun(int x){
   …
```

```
    fun(x-1);
    …
}
```

下列代码在 fun1( )函数中调用了 fun2( )函数，在 fun2( )函数中又调用了 fun1( )函数，因此 fun1( )函数间接地调用 fun1( )函数，也是递归调用。

```
int fun1(int x){
    …
    fun2(x-1);
    …
}
int fun2(int y){
    …
    fun1(y-1);
…
}
```

为了防止递归调用无终止地进行，必须在函数体中有终止递归调用的语句，称为“边界条件”。例如，使用条件判断语句，当满足某种条件时，就不再进行递归调用，这样递归调用能够逐层返回到最初的主调函数中。

使用递归函数时，递归的层次数（自己调用自己的次数）不能太大，否则会出现栈溢出而导致程序出错。

【例 7-8】 程序 7-7：函数递归调用示例，输入一个整数 *n*，求 *n* 的阶乘。

```
#01: //程序 7-7
#02: #include <stdio.h>
#03: long factorial(int);
#04:
#05: int main(){
#06:    int n;
#07:    long result;
#08:
#09:    printf("Input a integer:");
#10:    scanf("%d",&n);
#11:    result=factorial(n);
#12:    printf("%d!=%8ld\n",n,result);
#13:    return 0;
#14: }
#15: long factorial(int n){
#16:    long fac=0;
#17:    if(n<0)
#18:       printf("%d<0,data error!\n",n);
#19:    else if (n==0||n==1)
#20:       fac=1;
#21:    else
#22:       fac=n*factorial(n-1);
#23:    return (fac);
#24: }
```

程序解释：

#11：main( )函数调用函数 factorial( )，得到阶乘值。

#15～#24：factorial( )函数定义，该函数是一个递归函数。

#17, #18：判断传递的参数是否合法，如果小于 0，则非法。

#19, #20：如果传递的参数值为 0 或 1，则阶乘值为 1，结束该函数调用。

#21, #22：其他情况（参数值大于 1），则递归调用 factorial( )函数自身，并修改参数值为 n–1。这样，当参数的值减小到 1 时再递归调用，由于参数值为 1，终止递归调用。

函数调用关系如图 7-4 所示。

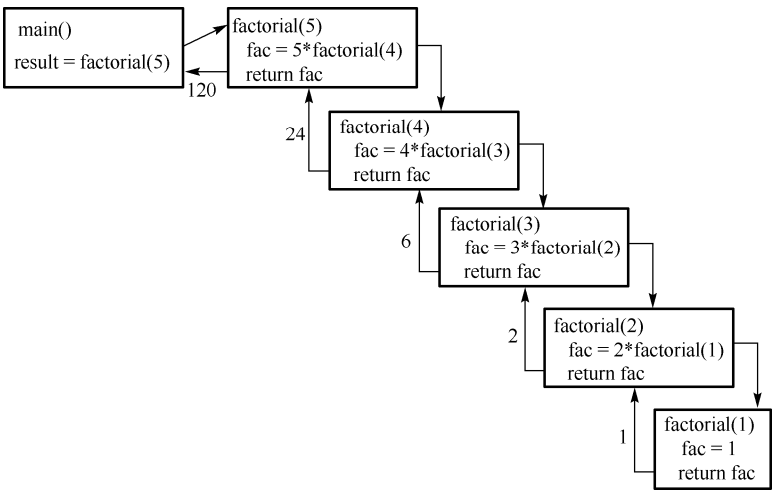


图 7-4　函数递归调用示例

程序运行结果如下：

```
Input a integer:5
5!=    120
```

7.4　数组与函数参数

7.4.1　函数参数传递方式

函数的实参可以是常量、变量或表达式等，类型必须与形参一致。实参的值有两类：①保存在内存中的值；②内存的地址（即指针）。因此实参将值传递给形参时也有两种方式：值传递和地址传递。

1．值传递

值传递方式是指：函数调用时，临时为形参分配内存单元，并将实参的值复制到形参的内存单元中；函数调用结束时，形参的内存单元被释放，但是实参的内存单元仍保留并维持原值。因此，形参与实参占用不同的内存单元，它们之间的传递是“单向传递”，即函数调用时实参值被单向传递给形参，被调用函数如果改变了形参的值，是不会影响实参的值的。

【例 7-9】 程序 7-8：值传递示例。

```
#01: //程序 7-8
```

```
#02: #include <stdio.h>
#03: void swap(int,int);
#04: int main(){
#05:    int a=1,b=2;
#06:
#07:    printf("a=%d,b=%d\n",a,b);
#08:    swap(a,b);
#09:    printf("a=%d,b=%d\n",a,b);
#10:    return 0;
#11: }
#12:
#13: void swap(int x, int y){
#14:      int t;
#15:      printf("x=%d,y=%d\n",x,y);
#16:      t=x;
#17:      x=y;
#18:      y=t;
#19:      printf("x=%d,y=%d\n",x,y);
#20: }
```

程序解释：

#07：调用 swap( ) 函数之前，a=1，b=2。

#08：调用 swap 函数时，将实参 a 和 b 的值单向传递给形参 x 和 y。

#09：在 swap( ) 函数中无论怎么改变形参值，实参 a 和 b 都不会改变。

#15：在 swap( ) 函数中，此时 x=1，y=2。

#16～#18：交换变量 x 和 y 的值，由于改变的是形参值，实参 a 和 b 的值不会改变。

#19：在 swap( ) 函数中，形参的值被改变了。

该程序实参和形参值的变化情况如图 7-5 所示。

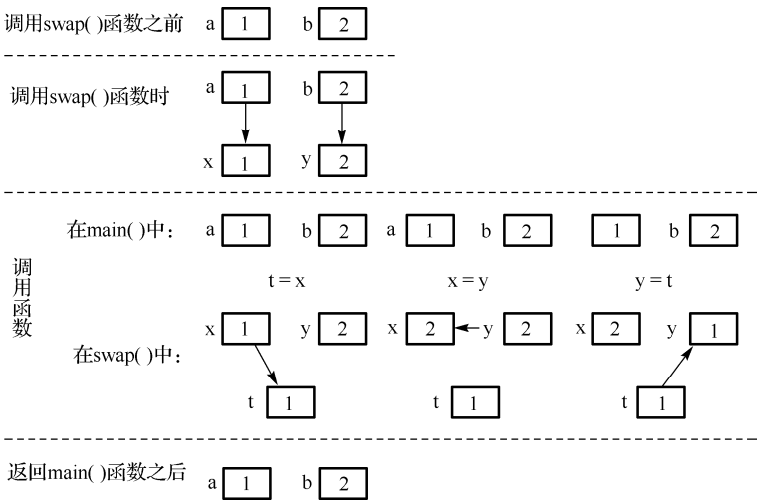


图 7-5 值传递示意图

程序运行结果如下：

```
a=1,b=2
x=1,y=2
x=2,y=1
a=1,b=2
```

2．地址传递

地址传递方式是指：实参存储的是内存地址，当函数调用时，实参将内存地址传递给形参，因此实参和形参公用同一个内存单元。虽然它们之间的传递仍然是“单向传递”，但形参变量和实参变量指向同一个内存单元，形参对该内存单元的修改会影响实参指向的值。

数组名和指针都可以作为实参，通过地址传递方式传递给形参。

【例 7-10】 程序 7-9：地址传递示例。

```
#01: //程序 7-9
#02: #include <stdio.h>
#03: void swap(int[]);
#04: int main(){
#05:     int a[2]={1,2};
#06:
#07:     printf("a[0]=%d,a[1]=%d\n",a[0],a[1]);
#08:     swap(a);
#09:     printf("a[0]=%d,a[1]=%d\n",a[0],a[1]);
#10:     return 0;
#11: }
#12:
#13: void swap(int x[]){
#14:       int t;
#15:       printf("x[0]=%d,x[1]=%d\n",x[0],x[1]);
#16:       t=x[0];
#17:       x[0]=x[1];
#18:       x[1]=t;
#19:       printf("x[0]=%d,x[1]=%d\n",x[0],x[1]);
#20: }
```

程序解释：

#07：调用 swap( )函数之前，a[0]=1，a[1]=2。

#08：调用 swap( )函数时，将实参 a 的值单向传递给形参 x。由于 a 是数组首元素的地址，因此 x 也指向数组首元素，即 a[0]。

#09：在 swap( )函数中改变了 x 指向的内存单元，由于 x 和 a 指向同一个内存单元，因此 a 指向的内存单元的值也改变了。

#13：定义 swap( )函数，形参“x[]”也可以写成“x[2]”，其中的“2”可有可无，没有实际意义。

#15：在 swap( )函数中，此时 x[0]=1，x[1]=2。

#16～#18：交换变量 x[0]和 x[1]的值，由于 x 和 a 指向同一个内存单元，因此等价于交换 a[0]和 a[1]的值。

#19：在 swap( )函数中，x[0]和 x[1]的值被改变了。

该程序实参和形参值的变化情况如图 7-6 所示。

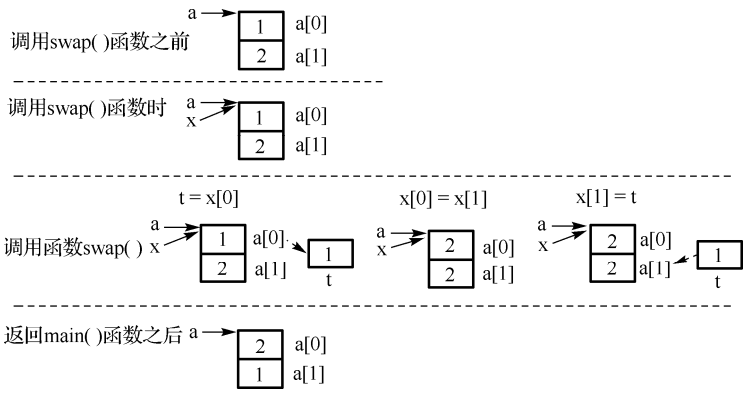


图 7-6 地址传递示意图

程序运行结果如下：

```
a[0]=1,a[1]=2
x[0]=1,x[1]=2
x[0]=2,x[1]=1
a[0]=2,a[1]=1
```

7.4.2 数组元素作为函数实参

数组作为一种函数实参进行数据传递有两种形式：①数组元素（数组下标变量）作为函数实参；②数组名作为函数形参和实参。

数组元素（数组下标变量）与普通变量没有区别，因此作为函数实参使用时与普通变量相同。对于形参，只要和数组的类型一致即可，不要求函数形参也是下标变量。主调函数调用函数时，把作为实参的数组元素的值传递给形参，实现单向的值传递。

【例 7-11】 程序 7-10：数组元素作为函数实参示例，输出数组中各元素的值，若大于 0，则输出该值，若小于等于 0，则输出该值的绝对值。

```
#01: //程序 7-10
#02: #include <stdio.h>
#03: #define LEN 5
#04: void output(int);
#05: int main(){
#06:     int a[LEN],i;
#07:
#08:     printf("Input %d numbers:",LEN);
#09:     for(i=0;i<LEN;i++)
#10:         scanf("%d",&a[i]);
#11:
#12:     printf("Array numbers:\n");
#13:     for(i=0;i<LEN;i++)
#14:         output(a[i]);
#15:     return 0;
#16: }
#17:
#18: void output(int val){
```

```
#19:    if(val>0)
#20:       printf("%d\t",val);
#21:    else
#22:       printf("%d\t",-val);
#23: }
```

程序解释：

#08～#10：输入 LEN 个整数。

#12～#14：按要求输出数组中的元素。

#14：调用 output( )函数，将变量 a[i]作为实参传递给函数形参。

#18～#23：函数 output( )的定义，对传递过来的值进行处理。

程序运行结果如下：

```
Input 5 numbers:-1 3 -5 -7 9
Array numbers:
1      3      5      7      9
```

7.4.3　数组名作为函数参数

数组名作为函数参数时，传递的是数组的首地址，是地址传递。

用数组名作为函数参数时，函数形参是数组，对应的实参必须是数组名。实参数组与形参数组的类型必须相同。虽然函数形参是数组，实际上形参数组并不存在，编译器不为形参数组分配内存，参数传递时只是地址传递，即把实参数组的首地址赋值给形参数组名，这样形参数组和实参数组为同一数组，共同拥有一段内存空间。

数组名作为函数参数传递时，形参数组大小（或二维数组第一维）可不指定，但需要另外一个参数传递数组的个数。

【例 7-12】 程序 7-11：数组名作为函数参数示例，输入一个整型数组，输出数组元素的平均值。

```
#01: //程序 7-11
#02: #include <stdio.h>
#03: #define LEN 5
#04: float average(int arr[], int n);
#05: int main(){
#06:    int array[LEN],i;
#07:    float  aver;
#08:
#09:    printf("Input %d intergers:",LEN);
#10:    for(i=0; i<LEN; i++ )
#11:       scanf("%d",&array[i]);
#12:
#13:    aver=average(array,LEN);
#14:    printf("Average of array:%.2f\n",aver);
#15:    return 0;
#16: }
#17:
#18: float average(int arr[], int n){
#19:    int i;
```

```
#20:    float av,total=0;
#21:
#22:    for(i=0;i<n;i++ )
#23:       total += arr[i];
#24:    av = total/n;
#25:
#26:    return av;
#27: }
```

程序解释：

#13：调用 average( )函数，实参为数组名 array 和数组长度 LEN。

#18：定义 average( )函数，形参为数组 arr[]，因为参数传递时只是地址传递，因此需要另外一个参数 n 作为传递的数组的长度。形参“arr[]”等价于“arr[LEN]”，其中“LEN”没有实际意义，仍然需要数组长度参数 n。

程序运行结果如下：

```
Input 5 intergers:1 11 111 1111 11111
Average of array:2469.00
```

当使用数组名作为参数传递时，形参和实参指向同一个数组，因此当形参数组发生变化时，实参数组也会随之变化，但是不能理解为发生了“双向”的值传递，传递过程始终是“单向”的，只是因为实参与形参指向的内存单元相同而使得实参数组的值随形参数组值的变化而变化。

数组名作为参数传递，只传送首地址而不检查形参数组的长度，因此形参数组和实参数组的长度可以不相同，这样对数组的处理可以很灵活。但是如果要求两者长度相同，但是传递的数组长度不一致时，可能会出现错误。

【例 7-13】 程序 7-12：将数组下标为 1、3、5…的数组元素值取相反数。

```
#01: //程序 7-12
#02: #include <stdio.h>
#03: #define LEN 6
#04: void change(int arr[], int n);
#05: int main(){
#06:    int array[LEN],i;
#07:    printf("Input %d intergers:",LEN);
#08:    for(i=0; i<LEN; i++ )
#09:       scanf("%d",&array[i]);
#10:
#11:    change(array,LEN/2);
#12:    printf("Changed array:");
#13:    for(i=0; i<LEN; i++)
#14:       printf("%3d",array[i]);
#15:
#16:    printf("\n");
#17:    return 0;
#18: }
#19:
#20: void change(int arr[], int num){
```

```
#21:    int i;
#22:    for(i=0;i<num;i++)
#23:       arr[2*i+1]=-arr[2*i+1];
#24: }
```

程序解释：

#11：调用 change( )函数，实参为数组名 array 和需要改变的数组元素个数。

#13, #14：调用 change( )函数，改变了奇数下标的元素值，实参对应的值也发生了改变。

#20：定义 change( )函数，形参为数组 arr[]和要改变的数组元素个数。

程序运行结果如下：

```
Input 6 intergers:1 2 3 4 5 6
Changed array:  1 -2  3 -4  5 -6
```

7.5　指针与函数参数

7.5.1　指针变量作为参数

可以使用指针变量作为函数参数，将一个变量或数组的地址传送到另一个变量中，因此指针变量作为函数参数是“地址传递”。通常在主调函数中使用指针变量指向一个变量或数组，然后将指针变量作为实参，把地址传递给调用函数的形参；通过形参指针变量，改变目标变量的值，在主调函数中就可以使用已改变了值的变量或数组。

【例 7-14】 程序 7-13：交换两个变量的值。

```
#01: //程序 7-13
#02: #include <stdio.h>
#03: void swap(int*,int*);
#04: int main(){
#05:    int a=1,b=2;
#06:    int *ptr1=&a,*ptr2=&b;
#07:
#08:    printf("a=%d,b=%d\n",a,b);
#09:    swap(ptr1,ptr2);
#10:    printf("a=%d,b=%d\n",a,b);
#11:    return 0;
#12: }
#13:
#14: void swap(int* p1,int* p2){
#15:      int t;
#16:      printf("*p1=%d,*p2=%d\n",*p1,*p2);
#17:      t=*p1;
#18:      *p1=*p2;
#19:      *p2=t;
#20:      printf("*p1=%d,*p2=%d\n",*p1,*p2);
#21: }
```

程序解释：

#06：指针变量 ptr1 和 ptr2 分别指向变量 a 和 b。

#08：调用 swap( )函数之前，a=1，b=2。

#09：调用 swap 函数时，将指针变量 ptr1 和 ptr2 的值单向传递给形参指针变量 p1 和 p2。由于 ptr1 和 ptr2 中为变量 a 和 b 的地址，因此 p1 和 p2 分别指向变量 a 和 b。

#10：在 swap 函数中改变了指针变量 p1 和 p2 指向的内存单元，即变量 a 和 b 的值。

#14：定义 swap( )函数，形参为指针变量。

#16：在 swap( )函数中，p1 和 p2 指向变量 a 和 b，因此分别输出 a 和 b 的值。

#17～#19：交换指针变量 p1 和 p2 指向内存单元的值，等价于交换 a 和 b 的值。

#20：在 swap( )函数中，a 和 b 的值被交换。

程序运行结果如下：

```
a=1,b=2
*p1=1,*p2=2
*p1=2,*p2=1
a=2,b=1
```

该程序实参和形参值的变化情况如图 7-7 所示。

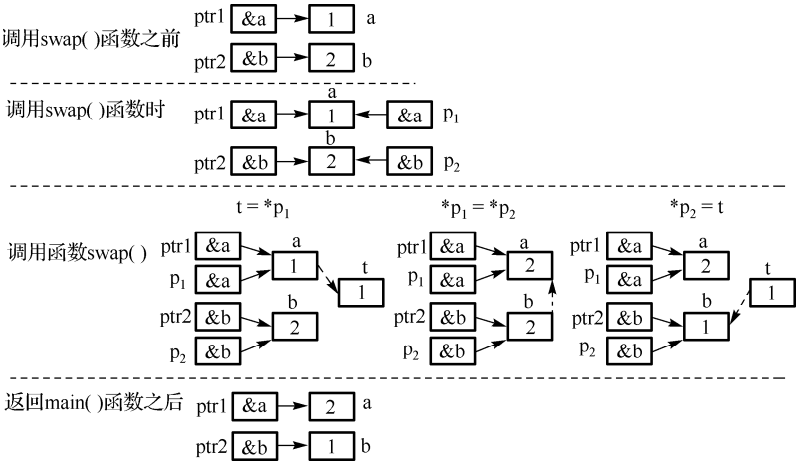


图 7-7 指针变量作为函数参数示意图

例 7-14 中的“swap( )”函数如果写成下列形式，则由于指针变量“*p”未指向任何变量的地址，而导致语句“*p=*p1;”运行时出现错误，因此指针变量在使用前必须先赋值。

```
void swap(int* p1,int* p2){
    int *p;
    *p=*p1;
    *p1=*p2;
    *p2=*p;
}
```

如果将“swap( )”函数写成下列形式，则函数调用时是“地址传递”，p1 和 p2 分别指向变量 a 和 b 的地址，但是该函数只是把指针变量 p1 和 p2 的值进行交换，即 p1 指向变量 b，而 p2 指向变量 a，并不改变目标变量的值，因此该函数无法完成将变量 a 和 b 交换的功能。

```
void swap(int* p1, int* p2){
   int *p;
   p=p1;
   p1=p2;
   p2=p;
}
```

如果将“swap( )”函数写成下列形式，而在 main( )函数中使用语句“swap(*ptr1,*ptr2);”调用 swap( )函数，则函数调用时是“值传递”。形参为两个 int 型变量，实参为指针变量 ptr1 和 ptr2 指向的目标变量的值，即变量 a 和 b 的值，因此该函数无法完成将变量 a 和 b 交换的功能。

```
void swap(int x,int y){
   int t;
   t=x;
   x=y;
   y=t;
}
```

通常将指针变量作为函数参数，以使函数“返回”多个值，本质上函数只有一个返回值，由于在被调用函数中，指针变量可以指向主调函数中的目标变量，通过修改指针变量指向的内存单元的值，就可以修改目标变量的值，这样将函数的处理结果“间接”地反映到主调函数中。

【例 7-15】 程序 7-14：输入一个整型数组，输出数组的最大值、最小值和平均值。

```
#01: //程序 7-14
#02: #include <stdio.h>
#03: #define LEN 5
#04: float find_element(int[],int,int*,int*);
#05: int main(){
#06:    int array[LEN],i,max,min;
#07:    float aver;
#08:
#09:    printf("Input %d intergers:",LEN);
#10:    for(i=0; i<LEN; i++ )
#11:       scanf("%d",&array[i]);
#12:    aver=find_element(array,LEN,&max,&min);
#13:    printf("Average of array:%.2f,max:%d,min:%d\n",aver,max,min);
#14:    return 0;
#15: }
#16:
#17: float find_element(int arr[],int n,int *pmax,int *pmin){
#18:    int i;
#19:    float av,total;
#20:
#21:    *pmax=arr[0];
#22:    *pmin=arr[0];
#23:    total=arr[0];
#24:    for(i=1;i<n;i++){
#25:       total += arr[i];
```

```
#26:        if (*pmax<arr[i])
#27:            *pmax=arr[i];
#28:        else if (*pmin>arr[i])
#29:            *pmin=arr[i];
#30:    }
#31:    av = total/n;
#32:
#33:    return av;
#34: }
```

程序解释：

#12：调用 find_element( )函数，实参为数组名 array、数组长度 LEN、变量 max 和 min 的地址。

#17：定义 find_element( )函数，形参为数组 arr[]、接收数组长度的变量 n、两个指针变量 pmax 和 pmin，用于改变主调函数中的目标变量 max 和 min。

#21, #22：初始化*pmax、*pmin 的值，即目标变量 max 和 min 的值为数组首元素。

#26～#29：将数组当前元素 arr[i]与*pmax 和*pmin 比较，根据比较结果修改 pmax 或 pmin 指向的内存单元，即变量 max 和 min。

程序运行结果如下：

```
Input 5 intergers:9 8 7 6 5
Average of array:7.00,max:9,min:5
```

7.5.2 指针变量和数组作为参数

由于数组名和指针变量作为函数参数都是“地址传递”，因此在函数调用中，数组名和指针变量可以等价使用。数组名作实参时，代表该数组首元素的地址，也可以使用指针变量作为实参。形参用来接收从实参传递过来的数组首元素的地址，因此形参应该是一个存放地址的指针变量或数组。具体地，实参与形参可以有 4 种：①数组名作为实参，数组作为形参；②数组名作为实参，指针变量作为形参；③指针变量作为实参，数组作为形参；④指针变量作为实参，指针变量作为形参。

对于函数 fun( )，形参的表示方法“fun(int arr[], int n)”与“fun(int *arr, int n)”是等价的。编译器将形参数组名作为指针变量来处理，调用函数时，系统会建立一个指针变量 arr，用来存放从主调函数传递过来的实参数组首元素的地址，这样，arr 指向实参数组首元素 array[0]。在函数内部，对于第 i 个数组元素，“*(arr+i)”和“arr[i]”两种表示方法是等价的。

【例 7-16】 程序 7-15：将数组中的元素按相反顺序存放。

```
#01: //程序 7-15
#02: #include <stdio.h>
#03: #define LEN 5
#04: void inverse(int[],int);
#05: int main(){
#06:    int array[LEN],i;
#07:
#08:    printf("Input %d intergers:",LEN);
#09:    for(i=0; i<LEN; i++ )
#10:        scanf("%d",&array[i]);
#11:
#12:    inverse(array,LEN);
```

```
#13:    printf("Inverse array:");
#14:    for(i=0; i<LEN; i++)
#15:        printf("%3d",array[i]);
#16:
#17:    printf("\n");
#18:    return 0;
#19: }
#20:
#21: void inverse(int arr[],int n){
#22:    int temp,i;
#23:
#24:    for(i=0;i<n/2;i++){
#25:        temp=arr[i];
#26:        arr[i]=arr[n-1-i];
#27:        arr[n-1-i]=temp;
#28:    }
#29: }
```

程序解释：

#12：调用 inverse( )函数，实参为数组名 array、数组长度 LEN。

#21：定义 inverse( )函数，形参为数组 arr[]、接收数组长度的变量 n。

#24～#28：以数组中间元素（下标为 n/2）为界线，将数组下标为 i 和 n–1–i 的两个元素对调位置。

程序运行结果如下：

```
Input 5 intergers:1 2 3 4 5
Inverse array:  5  4  3  2  1
```

例 7-16 中，如果数组名作为实参，指针变量作为形参，则 inverse( )函数可以改写为如下代码（main( )函数代码不变）。inverse( )函数中使用了三个指针变量 start、end、mid，分别指向数组的开始、结束和中间位置，对调指针 start 和 end 所指向的元素，并调整指针 start 和 end 向数组中间移动，从而实现将数组反转。

```
void inverse(int* ptr,int n){
    int temp,i;
    int *start,*end,*mid;

    start=ptr;
    end=ptr+n-1;
    mid=ptr+(n/2);

    for(;start<mid;start++,end--){
        temp=*start;
        *start=*end;
        *end=temp;
    }
}
```

例 7-16 中，如果指针变量作为实参，数组名作为形参，则 main( )函数可以改写为如下代码（inverse( )函数代码不变）。

```
int main(){
    int array[LEN],i;
    int *parray;

    parray=array;
    printf("Input %d intergers:",LEN);
    for(i=0; i<LEN; i++ )
        scanf("%d",parray++);

    parray=array;
    inverse(parray,LEN);
    printf("Inverse array:");
    for(i=0; i<LEN; i++)
        printf("%3d",*parray++);

    printf("\n");
    return 0;
}
```

例 7-16 中，如果指针变量作为实参，指针变量作为形参，则 inverse( )函数和 main( )函数都需要修改为上述代码。

7.6 变量种类及存储类型

7.6.1 变量种类

变量有效性的范围称为变量的作用域，C 语言中变量按作用域范围可分为两类：局部变量和全局变量。

1. 局部变量

凡在函数内部定义的变量称为局部变量，也称内部变量，其作用域仅限于所定义的函数内部，在该函数外部使用局部变量是非法的。例如，在 main( )函数中定义的变量只能在 main( )函数中有效。形参变量是属于被调用函数的局部变量，实参变量是属于主调函数的局部变量。例如，下列程序块中，变量 m 和 n 在 main( )函数中有效，变量 a、b 和 c 在 fun1( )函数中有效，变量 x、p、i 和 j 在 fun2( )函数中有效。

```
int main(){
    int m,n;
}
int fun1(int a) {
    int b,c;
}
float fun2(int x,int* p){
    int i,j;
}
```

在复合语句中，也可以定义仅在复合语句块中有效的临时变量，这种复合语句称为“分程序”或“程序块”。例如，下列程序块中的变量 temp 只在 for 循环体中有效。

```
for(i=0;i<n/2;i++){
    int temp;
    temp=arr[i];
    arr[i]=arr[n-1-i];
    arr[n-1-i]=temp;
}
```

局部变量的存储类型有 auto、register、static 和 auto 等，默认为 auto 类型，将在 7.6.2 节介绍。

不同的函数可有同名同类型的变量，它们占用不同的内存单元，并且互不影响。例如，主调函数的实参和被调用函数的形参的变量名可以相同，作用域不同。

【例 7-17】 程序 7-16：不同函数中可以有同名变量。

```
#01: //程序 7-16
#02: #include <stdio.h>
#03: fun1(){
#04:    int a=2;
#05:
#06:    {
#07:       int a=3;
#08:       printf("fun1 sub:a=%d\n",a);
#09:    }
#10:    printf("fun1:a=%d\n",a);
#11: }
#12: int main(){
#13:    int a=1;
#14:
#15:    fun1();
#16:    printf("main:a=%d\n",a);
#17:    return 0;
#18: }
```

程序解释：

#04：fun1( ) 函数中定义了一个局部变量 a，初始值为 2。

#07：fun1( ) 函数中的复合语句程序块中定义了一个局部变量 a，初始值为 3，它与#04 行的变量 a 虽然同名，但是占用不同的内存空间，作用域也不同。

#08：复合语句程序块中的局部变量 a 有效，此时“屏蔽”了#04 行定义的同名变量 a。

#10：fun1( ) 函数中的局部变量 a 有效。

#13：main( ) 函数中定义了一个局部变量 a，初始值为 1。

#16：main( ) 函数中的局部变量 a 有效。

程序运行结果如下：

```
fun1 sub:a=3
fun1:a=2
main:a=1
```

2．全局变量

在函数外部定义的变量称为全局变量，也称外部变量，其作用域是整个源程序，被本文件所有函数公用。全局变量从定义变量的位置开始，到本源文件结束都有效，甚至在有“extern”说明的其他源文件也可以有效。例如，下列程序块中，变量 sum 和 k 都是全局变量，变量 sum 在所有函数中都有效，而变量 k 在 main( )函数之后定义，因此只能在 fun1( )函数和 fun2( )函数中有效，在 main( )函数中无效。

```
int sum;
int main(){
    int m,n;
}
int k;
int fun1(int a) {
    int b,c;
}
float fun2(int x,int* p){
    int i,j;
}
```

在函数中使用全局变量，相当于是函数的公共变量，若一个函数修改了全局变量的值，则另一个函数在使用该变量的值时亦发生了改变，因此可以使用全局变量在函数之间传递数值。但是程序中应尽量减少使用全局变量，因为全局变量会导致函数依赖于外部定义的变量，减少了通用性。

【例 7-18】 程序 7-17：输入一个整型数组，输出数组的最大值和最小值。

```
#01: //程序 7-17
#02: #include <stdio.h>
#03: #define LEN 5
#04: int max,min;
#05: void find_element(int[],int);
#06: int main(){
#07:     int array[LEN],i;
#08:
#09:     printf("Input %d intergers:",LEN);
#10:     for(i=0; i<LEN; i++ )
#11:         scanf("%d",&array[i]);
#12:     find_element(array,LEN);
#13:     printf("Max:%d,min:%d\n",max,min);
#14:     return 0;
#15: }
#16:
#17: void find_element(int arr[],int n){
#18:     int i;
#19:
#20:     max=arr[0];
#21:     min=arr[0];
#22:     for(i=1;i<n;i++)
#23:         if (max<arr[i])
#24:             max=arr[i];
```

```
#25:        else if (min>arr[i])
#26:            min=arr[i];
#27:  }
```

程序解释：

#04：定义两个全局变量 max 和 min，从定义开始到程序末尾都有效。

#13：输出全局变量 max 和 min 的值（在 find_element( )函数中被修改）。

#23～#26：find_element( )函数中，根据不同的条件对全局变量 max 和 min 赋值。

程序运行结果如下：

```
Input 5 intergers:3 4 5 1 2
Max:5,min:1
```

如果在同一个源文件中，有相同名字的全局变量和局部变量，则在局部变量的作用域内，全局变量被“屏蔽”，不起作用。

【例 7-19】 程序 7-18：局部变量屏蔽全局变量示例。

```
#01: //程序 7-18
#02: #include <stdio.h>
#03: int a=1,b=2;
#04: int max(int x,int y){
#05:    return(x>y?x:y);
#06: }
#07:
#08: int main() {
#09:    int a=3;
#10:
#11:    printf("Max is:%d.\n",max(a,b));
#12:    return 0;
#13: }
```

程序解释：

#03：定义两个全局变量 a 和 b 并赋初值。

#09：在 main( ) 函数中定义一个局部变量 a，赋初值 3。

#11：main( ) 函数中的局部变量 a“屏蔽”了同名的全局变量 a，因此函数调用“max(a,b)”等价于“max(3,2)”。

程序运行结果如下：

```
Max is:3.
```

7.6.2　存储类型

C 语言中的变量有两个属性：数据类型和存储类型，变量定义的一般形式为“[存储类型]数据类型 变量名,…;”。存储类型是指变量在内存中存储的方式。编译器提供一定的内存空间供程序使用，称为用户区。用户区被划分为不同的区域以存放不同种类的数据。一般来说，用户区分为三部分：①代码区，存放程序代码；②动态区，存放暂时的数据；③静态区，存放相对永久的数据。如图 7-8 所示。

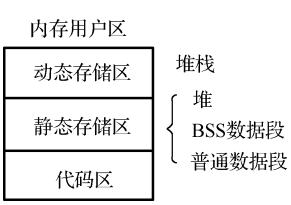


图 7-8　用户存储空间分配

1．动态存储与静态存储

变量按其存在的时间（生存期），可分为两种方式：静态存储和动态存储。静态存储指在程序运行期间分配固定的存储空间，动态存储指在程序运行期间根据需要动态地分配存储空间。

静态存储的变量存放在静态存储区，该区域存放全局变量、常量及使用 malloc( )等函数进行动态分配的数据。该区域可分为三部分：①堆，由用户通过函数（malloc( )/free( )等）分配和释放，若用户不释放，程序结束时由操作系统回收；②BSS（Block Started by Symbol）数据段，存放未初始化的全局变量；③普通数据段，存放静态变量、初始化的全局变量或常量。

动态存储的变量存放在动态存储区，也称“堆栈段”，该区域存放函数的返回地址、函数的参数（形参）、局部变量等。

静态存储的变量在程序编译时被分配存储单元，在程序执行过程中占据固定的存储空间，程序执行完后释放。动态存储的变量在函数开始调用执行时，由编译器自动分配动态存储空间，函数结束时自动释放这些空间。

2．局部变量存储类型

对于局部变量，其存储类型有三种：①自动变量；②寄存器变量；③静态局部变量。

（1）自动变量

自动变量存储在动态存储区，使用关键字“auto”说明，可以省略。定义变量时如果不写“auto”则隐含为自动变量。例如，下列程序块中，形参变量 a、局部变量 b 和 c 都是自动变量，变量 ch 省略了 auto 说明，也是自动变量。调用 fun( )函数时，系统自动为它们分配存储空间，函数调用结束时，自动释放存储空间。

```
int fun(int a) {
    auto int b,c;
    char ch;
}
```

（2）寄存器变量

为了提高程序运行效率，可以将局部变量的值放在 CPU 的寄存器中，这种变量称为“寄存器变量”，使用关键字“register”说明，寄存器变量对寄存器的占用是动态的。只有自动变量和形参变量可以作为寄存器变量。

CPU 的寄存器数量有限，不能定义很多寄存器变量，有的系统将寄存器变量转化为自动变量。

例如，下列程序块中，定义了两个寄存器变量 i 和 fac。

```
int factorial(int n){
    register int i,fac=1;
    for(i=1;i<=n;i++){
        fac=fac*i;
    }
    return (fac);
}
```

（3）静态局部变量

如果希望函数中的局部变量的值在函数调用结束后不消失而继续保留当前值，可以指定该局部变量为静态局部变量，使用关键字“static”说明。

静态局部变量与自动变量有如下不同。

①静态局部变量属于静态存储，存放在静态存储区，在程序整个运行期间都不会释放存储空间；而自动变量属于动态存储，存放在动态存储空间，在函数调用结束时释放存储空间。

②静态局部变量在编译时赋初值，仅赋初值一次；而自动变量在函数调用时赋初值，每调用函数一次就给自动变量赋一次初值，相当于执行了一次赋值语句。

③如果定义局部静态变量时没有赋初值，编译时会自动赋初值 0（对数值型变量）或空字符（对字符变量）；而对自动变量，如果不赋初值，则它的值是一个不确定的值。

使用静态局部变量时，应注意：

①静态局部变量只能在定义它的函数内部使用，虽然在函数外部它仍然存在，但是不能被其他函数使用；

②对静态局部变量，若定义时赋初值，则在程序运行中仅第一次调用函数时赋初值，下一次调用不再赋初值，而是使用上一次调用的值。

【例 7-20】 程序 7-19：静态局部变量示例。

```
#01: //程序 7-19
#02: #include <stdio.h>
#03: void func(){
#04:     static int x=0;
#05:     int y=0;
#06:     printf("x=%d,y=%d\n",x,y);
#07:     x++;
#08:     y++;
#09: }
#10:
#11: int main() {
#12:     func();
#13:     func();
#14:     func();
#15:     return 0;
#16: }
```

程序解释：

#04：定义一个静态局部变量 x，在对 func( )函数的三次调用中，只有第一次调用会对变量 x 进行初始化。

#05：定义一个自动变量 y，在对 func( )函数的三次调用中，每次调用都会对变量 y 进行初始化。

#07, #08：输出变量 x 和 y 的值后，修改变量 x 和 y。

#12：第一次调用时，输出变量 x 值为 0，y 值为 0，此后变量 x 和 y 值都变为 2，函数调用结束后，变量 x 存储空间仍然存在，变量 y 存储空间被释放。

#13：第二次调用时，输出变量 x 值为 1，y 值为 0，因为变量 x 保持上次调用时的值，而变量 y 重新分配存储空间，重新初始化。

#14：第三次调用时，输出变量 x 值为 2，y 值为 0，变量 x 仍然保持上次调用时的值，而变量 y 仍然重新分配存储空间，重新初始化。

程序运行结果如下：

```
x=0,y=0
```

```
x=1,y=0
x=2,y=0
```

3．全局变量有效范围

全局变量属于静态存储，存放在静态存储区。除了一般全局变量，全局变量还可以有两种有效范围说明：①使用 static 说明；②使用 extern 说明。

（1）用关键字 static 说明的全局变量，只能在本源文件中使用，不能在其他源文件中被引用，其他源文件中可以定义相同名字的全局变量，不会产生冲突。例如，下列程序块中，全局变量 sum 只能在本源文件中有效。

```
static int sum;
int main(){
}
int fun() {
}
```

（2）用关键字 extern 说明的全局变量，说明该全局变量是在本源文件后面定义的，或者在其他源文件中定义的，一般形式为“extern 数据类型 变量表;”。例如，下列程序块中，虽然全局变量 k 在 fun1( ) 函数后面定义，但是在文件开始使用“exern int k;”语句，将全局变量 k 的作用域扩展到了整个源文件。

```
exern int k;
int main(){
}
int fun1(int a) {
}
int k;
float fun2(int x,int* p){
}
```

下列程序块中，“exern int k;”语句在 fun1( )函数内，因此全局变量 k 的作用域被扩展到 fun1( ) 函数中。

```
int main(){
}
int fun1(int a) {
    exern int k;
}
int k;
float fun2(int x,int* p){
}
```

下列程序块中，源文件 a.c 中定义了一个全局变量 sum，在源文件 b.c 中使用“extern int sum;”语句后，可以使用 a.c 中定义的全局变量 sum。

```
a.c:
int sum
int main(){
}
int fun(int a) {
```

```
    exern int k;
}

b.c:
extern int sum;
float fun2(){
}
```

7.7　程 序 示 例

【例 7-21】 程序 7-20：数制转换，输入一个 8 位二进制数，将其转换为十进制数输出。例如，输入 11101101，则输出 237。

```
#01: //程序 7-20
#02: #include <stdio.h>
#03: int power(int,int);
#04: int main(){
#05:    int  i,value=0;
#06:    char  ch;
#07:
#08:    printf("Input an 8 bits binary number:");
#09:    for (i=7;i>-1;i--){
#10:       ch=getchar();
#11:       if (ch=='\n'){
#12:          printf("Not enough 8 bits.\n");
#13:          return -1;
#14:       }
#15:
#16:       if (ch=='1')
#17:          value += power(2,i);
#18:    }
#19:    printf("Decimal value is:%d\n",value);
#20:    return 0;
#21: }
#22:
#23: int power(int base, int n){
#24:    int val =1;
#25:
#26:    while (n--)
#27:      val *= base;
#28:    return (val);
#29: }
```

程序解释：

#09～#18：根据输入的二进制数值，计算十进制数。要求输入 8 位。

#11～#14：如果输入的位数不足 8 位，则输出提示信息后程序退出。

#16, #17：根据二进制数到十进制数的转换规则，如果某一位为 1，则乘以位权 2^i，位权通过 power( ) 函数调用计算。

#23～#29：计算位权，形参为基数 2 和当前的位次顺序（从右向左）。

程序运行结果如下：

```
Input an 8 bits binary number:11101101
Decimal value is:237
```

【例 7-22】 程序 7-21：选择排序（Selection Sort）是一种简单直观的排序算法，工作原理如下：首先在未排序的序列中找到最小元素，存放到排序序列的起始位置；然后，再从剩余未排序的元素中继续寻找最小元素，然后放到已排序序列的末尾；以此类推，直到所有元素均排序完毕。

```
#01: //程序 7-21
#02: #include <stdio.h>
#03: #define LEN 5
#04: void sort(int*,int);
#05: int main(){
#06:     int array[LEN],i;
#07:
#08:     printf("Input %d integers:",LEN);
#09:     for(i=0;i<LEN;i++)
#10:         scanf("%d",&array[i]);
#11:
#12:     sort(array,LEN);
#13:     printf("Sorted array:");
#14:     for(i=0;i<LEN;i++)
#15:         printf("%3d",array[i]);
#16:     printf("\n");
#17:     return 0;
#18: }
#19:
#20: void sort(int *ptr,int num){
#21:     int i,j,min,temp;
#22:
#23:     for(i=0;i<num-1;i++){
#24:         min=i;
#25:         for(j=i+1;j<num;j++)
#26:             if (*(ptr+min)>*(ptr+j))
#27:                 min=j;
#28:
#29:         if (min!=i){
#30:             temp=*(ptr+min);
#31:             *(ptr+min)=*(ptr+i);
#32:             *(ptr+i)=temp;
#33:         }
#34:     }
#35: }
```

程序解释：

#12：调用 sort( ) 函数进行选择排序。

#20：定义 sort( )函数，形参为数组首元素地址和数组长度。

#24：首先假设最小元素位置为 i。

#25～#27：从 i+1 位置到 num–1 位置，寻找比 min 位置更小的元素，将其位置赋值给 min。

#29～#33：如果找到了比 i 位置更小的元素，则交换 i 位置和 min 位置的元素。

程序运行结果如下：

```
Input 5 integers:5 4 3 2 1
Sorted array:  1  2  3  4  5
```

上机实验：函数程序设计应用

本次实验掌握 C 语言程序的函数定义与调用，熟练掌握函数的定义方法、函数的调用方法；掌握主调函数和被调用函数之间的参数传递方式。

（1）输入一个整数，判断是否为素数。

程序示例：

```
#include <stdio.h>
#include <math.h>
int prime(int num){
   int i,flag=1;
   for(i=2;i<=sqrt(num);i++)
      if (num%i==0){
         flag=0;
         break;
      }

   return  (flag);
}

int main() {
   int number;

   printf("Input a number:");
   scanf("%d",&number);

   if (prime(number))
      printf("%d is a prime\n",number);
   else
      printf("%d is NOT a prime\n",number);
   return 0;
}
```

（2）比较两个数组大小。假设 a 和 b 为 N 个元素的整型数组，比较两个数组对应元素的大小，用变量 m 和 n 记录 a[i]>b[i]和 a[i]<b[i]的个数。如果 m>n，则数组 a>b；如果 m <n，则数组 a<b；如果 m ==n，则数组 a==b。

程序示例：

```
#include <stdio.h>
#define LEN 5
void compare(int*,int*,int);
int m=0,n=0;
int main(){
   int a[LEN],b[LEN],i;

   printf("Input %d elements array a:",LEN);
   for(i=0;i<LEN;i++)
      scanf("%d",&a[i]);
   printf("Input %d elements array b:",LEN);
   for(i=0;i<LEN;i++)
      scanf("%d",&b[i]);

   compare(a,b,LEN);

   if (m>n)
      printf("Array a > array b\n");
   else if (m<n)
      printf("Array a < array b\n");
   else
      printf("Array a == array b\n");

   return 0;
}

void compare(int* pa,int* pb,int num){
   int i;
   for(i=0;i<num;i++)
      if (*(pa+i)>*(pb+i))
         m++;
      else if (*(pa+i)<*(pb+i))
         n++;
}
```

习　题

1．编写求圆面积的函数，并调用该函数求出圆环的面积。

2．编写求两个数中最大数的函数，并调用该函数求出三个数中的最大数。

3．编写求 $k!$ 的函数，并调用该函数求 1!+2!+3!+…+6! 并输出。

4．编写求 $k!$ 的函数，并调用该函数求 $C(m,n)=m!/(n!\ (m-n)!)$ 并输出。

5．编写判定闰年的函数，并调用该函数判定某一年是否为闰年。

6．编写一个将实数四舍五入到小数点后第 n 位的函数，并调用此函数将一个实数舍入到小数点后第 2 位（内部精度而非输出精度）。

7．用函数嵌套调用实现：输入三个数，求其中最大数和最小数的差值。

8．编写函数将给定的一个二维数组（4×4）转置，即行列互换。

9．编写求 n 个数平均值的函数，并调用该函数找出长度为 n 的数组中小于平均值的元素。

10．用指针变量作为函数参数实现：输入三个整数，按大小顺序输出。

11．用静态局部变量方法，输出 1 到 5 的阶乘值。

12．用函数实现求 π 的值，计算 π 的公式为：$\pi = 16\arctan\left(\frac{1}{5}\right) - 4\arctan\left(\frac{1}{239}\right)$，其中 arctan( )函数用如下级数计算：$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5}\frac{x^7}{7} + \cdots$，直到级数某项绝对值不大于 10^{-15} 为止。

13．各位数字左右对称的整数称为回文数，寻找并输出 11～999 之间的数 m，它满足 m、m^2 和 m^3 均为回文数。

14．用递归方法计算从 n 个人中选择 k 个人组成一个委员会的不同组合数。提示：由 n 个人中选 k 个人的组合数=由 $n-1$ 个人里选 k 个人的组合数 + 由 $n-1$ 个人里选 $k-1$ 个人的组合数。当 $n=k$ 或 $k=0$ 时，组合数为 1。

15．编写求两个整数最大公约数的函数，并调用该函数求两个整数的最大公约数和最小公倍数。

提示：辗转相除法求最大公约数：

（1）以其中一个数作被除数，另一个数作除数，相除求余数；

（2）若余数不为 0，则将上一次的除数作为新的被除数，将上一次的余数作为新的除数，继续求余数；

（3）直至余数为 0 时，对应的除数就是最大公约数。

16．编写求字符串长度的函数，并调用该函数求一个字符串的长度。

17．编写字符串复制的函数，并调用该函数复制一个字符串。

18．编写连接两个字符串的函数，并调用该函数连接两个字符串。

19．编写将字符数组中的字符串前后倒置的函数，并调用该函数将一个字符串前后倒置。

20．编写判断回文的函数，并调用该函数判定一个字符串是否为回文。

第 8 章　构造数据类型

8.1　结　构　体

8.1.1　结构体类型

1．结构体变量定义

程序设计时，有时需要将一组不同类型的数据组合在一起，如学生信息，学号为整型、姓名为字符串、年龄为整型、性别为字符型、成绩为实数型，此时，可以使用结构体来表示和处理学生信息。

结构体是一种构造数据类型，是由一系列具有相同类型或不同类型的数据组成的数据集合。结构体由若干元素组成，这些元素称为结构体的成员（member），每个成员可以是一个基本数据类型，也可以是另外一个构造数据类型。

使用结构体通常先定义结构体类型，然后定义该类型的结构体变量。定义结构体类型的一般形式为：

```
struct 结构体名{
    成员列表
};
```

“成员列表”由若干成员组成，每个成员都需要说明类型，其一般形式为：“类型标识符 成员名;”。例如，下列程序块定义了一个 student 类型的结构体，共有 5 个成员。定义了一种结构体类型后可定义结构体变量，定义为 student 类型的结构体变量都有 5 个数据成员。

```
struct student{
    int id;
    char name[32];
    int age;
    char sex;
    float score;
};
```

结构体名和结构体变量是不同的，结构体名表示一个结构体类型，编译器不对它分配内存空间，只有当某个变量被定义为这种类型的结构体时，才对该变量分配存储空间。

定义结构体变量有三种方式：①先定义结构体类型，再定义结构体变量；②在定义结构体类型的同时定义结构体变量；③不定义结构体类型，直接定义结构体变量。

（1）先定义一个结构体类型，然后定义该类型的结构体变量。例如，定义了 student 类型的结构体后，可定义结构体变量 stu1：“struct student stu1;”，该变量 stu1 是一个 student 结构体类型的变量，有 5 个数据成员。

也可以一次定义多个结构体变量，如“struct student stu1, stu2;”。

（2）在定义结构体类型的同时定义一个或多个结构体变量，即将结构体变量名列表放在结构体类型定义之后（“}”和最后的“;”之间）。例如，下列程序块定义了两个 student 类型的结构体变量 stu1 和 stu2。

```
struct student{
    int id;
    char name[32];
    int age;
    char sex;
    float score;
}stu1, stu2;
```

（3）不定义结构体类型名称而直接定义一个或多个结构体变量，与第 2 种方法的区别在于省去了结构体名称，直接定义了结构体变量。例如，下列程序块定义了两个结构体变量 stu1 和 stu2。

```
struct{
    int id;
    char name[32];
    int age;
    char sex;
    float score;
}stu1, stu2;
```

以上三种定义方法是等价的，定义了结构体变量 stu1 和 stu2 之后，它们都具有图 8-1 所示的存储结构。

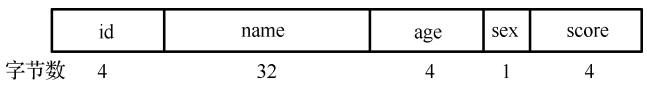


图 8-1　结构体存储结构 1

结构体成员也可以是另外一个结构体，形成嵌套的结构体。例如，下列程序块定义了两个结构体类型 data 和 student，其中 student 的成员变量 birthday 为 date 结构体类型，此时定义的结构体变量 stu1 和 stu2 都具有图 8-2 所示的存储结构。

```
struct date{
    int year;
    int month;
    int day;
};
struct{
    int id;
    char name[32];
    int age;
    char sex;
    struct data birthday;
    float score;
}stu1, stu2;
```

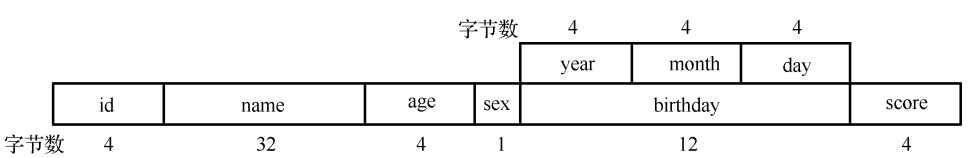


图 8-2　结构体存储结构 2

2．结构体变量初始化

与其他类型的变量一样，结构体变量可以在定义时进行初始化赋值。初始化时用花括号“{}”将初值常量列表括起来，用逗号“,”隔开，这些初值按顺序依次赋值给结构体变量的各个成员。

例如，下列程序块对结构体变量 stu1 进行初始化，初始化后成员 id=1、成员 name 为"Zhang3"、成员 age=21、成员 sex 为 'F '、成员 score=99。

```
struct student{
    int id;
    char name[32];
    int age;
    char sex;
    float score;
}stu1={1, “Zhang3”,21, ‘F’,99};
```

3．结构体变量引用

引用结构体变量成员的一般形式为：“结构体变量名.成员名”。其中，“.”是成员运算符。例如，“stu1.id”为变量 stu1 的 id 成员、“stu2.age”为变量 stu2 的 age 成员。如果成员同时又是另一个结构体类型，则使用若干成员运算符，一级一级地找到最低级的成员，如“stu1.birthday.day”。

使用了成员运算符的结构体成员与普通变量完全相同，可以在程序中独立使用，进行各种运算。

同类型的结构体变量可以相互赋值，但是结构体变量不能整体进行输入和输出，只能对结构体变量中的各个成员分别进行输入和输出。

【例 8-1】 程序 8-1：结构体变量示例。

```
#01: //程序 8-1
#02: #include <stdio.h>
#03: struct student{
#04:     int id;
#05:     char name[32];
#06:     int age;
#07:     char sex;
#08:     float score;
#09: };
#10: int main() {
#11:     struct student stu1,stu2;
#12:
#13:     stu1.id=101;
#14:     stu1.age=21;
#15:     stu1.score=99;
#16:     printf("Input sex[F/M] and name:");
#17:     scanf("%c%s",&stu1.sex,stu1.name);
#18:
#19:     stu2=stu1;
#20:     printf("stu2 info:\n");
#21:     printf("id=%d,name=%s,age=%d,sex=%c,score=%.1f\n",
                stu2.id,stu2.name,stu2.age,stu2.sex,stu2.score);
```

```
#22:     return 0;
#23: }
```

程序解释：

#13～#15：使用赋值语句给 id、age 和 score 成员赋值。结构体成员赋值顺序可以与定义结构体时的顺序不同。

#17：使用 scanf( )函数输入 sex 和 name 成员的值。

#19：直接对同类型的 stu2 进行赋值，将 stu1 的所有成员的值整体赋值给 stu2。

#21：使用结构体成员方法输出结构体变量。

程序运行结果如下：

```
Input sex[F/M] and name:F zhang3
stu2 info:
id=101,name=zhang3,age=21,sex=F,score=99.0
```

8.1.2　结构体数组

数组元素的类型可以是结构体类型，经常用结构体数组表示具有相同数据结构的一个群体，如一个学校所有学生的信息，结构体数组的每个元素都是具有相同结构体类型的数据。

定义结构体数组的一般形式为：

struct 结构体名{
　　成员列表
} 数组名[数组长度];

或者先定义一个结构体类型，然后再用该结构体类型定义一个结构体数组，一般形式为：

struct 结构体名{
　　成员列表
};
struct 结构体名 数组名[数组长度];

例如，下列程序块定义了一个 student 类型的结构体数组 stu，有 5 个数组元素，每个元素都是一个 student 类型的结构体。

```
struct student{
    int id;
    char name[32];
    int age;
    char sex;
    float score;
}stu[5];
```

对结构体数组可以进行初始化赋值，初始化方法是在定义数组的后面加上“={初值列表}”。当对全部数组元素都进行初始化赋值时，可以不给出数组长度。例如，下列程序块定义了一个 data 类型的结构体数组 holiday，有 3 个元素（数组长度为 3）。

```
struct date{
    int year;
```

```
    int month;
    int day;
}holiday[]={{2015,1,1},{2015,2,19},{2015,5,1}};
```

【例 8-2】 程序 8-2：定义一个学生结构体，计算 *N* 个学生的平均成绩和不及格的人数。

```
#01: //程序 8-2
#02: #include <stdio.h>
#03: #define LEN 5
#04: struct student{
#05:    int id;
#06:    char* name;
#07:    float score;
#08: };
#09: int main() {
#10:    int count=0,i;
#11:    float average=0;
#12:    struct student stu[LEN]={{1,"Zhang3",85},
#13:                             {2,"Li4",98},
#14:                             {3,"Wang5",56},
#15:                             {4,"Zhao6",78},
#16:                             {5,"Sun7",46}};
#17:
#18:    for(i=0;i<LEN;i++){
#19:       average+=stu[i].score;
#20:       if(stu[i].score<60)
#21:          count++;
#22:    }
#23:    average/=LEN;
#24:
#25:    printf("Average:%.1f\n",average);
#26:    printf("Score failed count:%d\n",count);
#27:    return 0;
#28: }
```

程序解释：

#04～#08：定义一个 student 类型的结构体。

#12～#16：定义一个 student 结构体类型的数组 stu，并进行初始化。可以组织在同一行。

#18～#23：计算平均值和分数小于 60 的个数。变量 average 在#19 行先用来保存成绩总数，然后在#23 行计算平均值。

程序运行结果如下：

```
Average:72.6
Score failed count:2
```

8.1.3 结构体指针

1. 结构体指针变量

结构体指针变量是指一个指针变量指向一个结构体变量，即结构体指针变量中存放了所指向的

结构体变量的首地址。与其他指针变量相同，通过结构体指针变量可以访问指向的目标变量（结构体变量）。

定义结构体指针变量的一般形式为：“struct 结构体名称 *结构体指针变量名;”，例如，定义了一个 student 类型的结构体后，可以使用语句“struct student *pstu;”定义一个 student 类型的结构体指针变量 pstu。也可以在定义 student 类型结构体的同时定义结构体指针变量 pstu，例如，下列程序块定义了一个 student 类型的结构体变量和一个 student 类型的结构体指针变量 pstu。

```
struct student{
    int id;
    char name[32];
    int age;
    char sex;
    float score;
}stu1,*pstu;
```

结构体指针变量必须先赋值然后才能使用，赋值是把一个结构体变量的首地址赋给该指针变量。例如，以上程序块中定义 stu1 和 pstu 后，语句“pstu=&stu1;”完成对指针变量 pstu 的赋值。

通过结构体指针变量访问结构体变量各个成员的一般形式为：“(*结构体指针变量).成员名”或“结构体指针变量→成员名”，它们与“结构体变量.成员名”是等价的。如“(*pstu).score”或“pstu→score”都可以访问结构体变量 stu1 的 score 成员。注意“(*pstu).score”形式中需要括号“( )”，因为成员运算符“.”优先级高于“*”，如果没有括号“( )”，则“*pstu.score”等价于“*(pstu.score)”。

【例 8-3】 程序 8-3：结构体指针变量示例。

```
#01: //程序 8-3
#02: #include <stdio.h>
#03: struct student{
#04:     int id;
#05:     char name[32];
#06:     float score;
#07: };
#08: int main() {
#09:     struct student stu,*pstu;
#10:     pstu=&stu;
#11:
#12:     printf("Input student id:");
#13:     scanf("%d",&(pstu->id));
#14:     fflush(stdin);
#15:     printf("Input student name:");
#16:     gets((*pstu).name);
#17:     printf("Input student score:");
#18:     scanf("%f",&stu.score);
#19:
#20:     printf("Student id:%d\n",stu.id);
#21:     printf("Student name:%s\n",pstu->name);
#22:     printf("Student score:%.1f\n",(*pstu).score);
#23:     return 0;
#24: }
```

程序解释：

#09：定义结构体变量 stu 和结构体指针变量 pstu。

#10：使结构体指针变量 pstu 指向结构体变量 stu。

#13, #21：使用“结构体指针变量→成员名”方式访问成员。

#14：清除#13 中输入的回车字符，以免影响后续输入。

#16, #22：使用“(*结构体指针变量).成员名”方式访问成员。

#18, #20：使用“结构体变量.成员名”方式访问成员。“&stu.score”等价于“&(stu.score)”，因为“.”运算符的优先级比“&”高，因此可以不加括号“( )”。

程序运行结果如下：

```
Input student id:1501001
Input student name:Zhang San
Input student score:92.5
Student id:1501001
Student name:Zhang San
Student score:92.5
```

2．结构体数组指针变量

结构体指针变量可以指向一个结构体数组，此时结构体指针变量的值是结构体数组第 0 个元素的首地址。可以移动结构体指针变量指向结构体数组中的其他元素，此时结构体指针变量的值是结构体数组其他元素的首地址。例如，如果结构体指针变量 ps 指向结构体数组第 0 个元素，则 ps+i 指向结构体数组的第 i 个元素。

结构体指针变量只能指向结构体数组中的元素，不能指向结构体数组元素中的成员。

【例 8-4】 程序 8-4：结构体数组指针变量示例。

```
#01: //程序 8-4
#02: #include <stdio.h>
#03: #define LEN 5
#04: struct student{
#05:     int id;
#06:     char* name;
#07:     float score;
#08: };
#09: int main() {
#10:     struct student *pstu;
#11:     struct student stu[LEN]={{1,"Zhang3",85},
#12:                              {2,"Li4",98},
#13:                              {3,"Wang5",56},
#14:                              {4,"Zhao6",78},
#15:                              {5,"Sun7",46}};
#16:
#17:     for(pstu=stu;pstu<stu+LEN;pstu++){
#18:         printf("Student %d info:\n",pstu-stu);
#19:         printf("\tid:%d\n",pstu->id);
#20:         printf("\tname:%s\n",pstu->name);
#21:         printf("\tscore:%.1f\n",pstu->score);
```

```
#22:    }
#23:    return 0;
#24: }
```

程序解释：

#10：定义结构体指针变量 pstu。

#17～#22：通过结构体指针变量 pstu 输出结构体数组 stu 的每个元素的成员。

#17：通过 pstu++移动指针变量 pstu，指向结构体数组的下一个元素。

#18：通过“pstu–stu”运算可以计算出 pstu 当前指向结构体数组的哪个元素。

程序运行结果如下：

```
Student 0 info:
   id:1
   name:Zhang3
   score:85.0
Student 1 info:
   id:2
   name:Li4
   score:98.0
Student 2 info:
   id:3
   name:Wang5
   score:56.0
Student 3 info:
   id:4
   name:Zhao6
   score:78.0
Student 4 info:
   id:5
   name:Sun7
   score:46.0
```

8.1.4　结构体与函数

结构体变量的成员、结构体变量和结构体指针变量都可以作为函数的参数。结构体变量的成员作为函数实参传递给形参，属于“值传递”方式，其用法和普通变量相同。

结构体变量和结构体指针变量可以作为函数参数进行整个结构体的传递。

1．结构体变量作为函数参数

结构体变量作为实参传递给形参，属于“值传递”方式，将结构体变量的所有成员按顺序传递给形参，形参要求是相同类型的结构体变量。函数调用时，需要为形参分配结构体大小的内存空间。因此结构体变量作为函数参数，会使时间和空间开销较大，降低了程序效率。此外，如果被调用函数修改了形参结构体的成员，该值不能返回到主调函数中。

【例 8-5】 程序 8-5：结构体变量作为函数参数示例。

```
#01: //程序 8-5
#02: #include <stdio.h>
```

```
#03: struct student{
#04:     int id;
#05:     char* name;
#06:     float score;
#07:     char flag;
#08: };
#09: void investigate(struct student);
#10: int main() {
#11:     struct student stu={15001,"Zhang3",52,'x'};
#12:
#13:     printf("Student flag:%c\n",stu.flag);
#14:     investigate(stu);
#15:     printf("Student flag:%c\n",stu.flag);
#16:     return 0;
#17: }
#18:
#19: void investigate(struct student var){
#20:     if (var.score<60){
#21:         var.flag='F';
#22:         printf("Warning:%s's score is %.1f < 60\n",var.name,var.score);
#23:     }
#24: }
```

程序解释：

#13：调用 investigate( ) 函数前，变量 stu 的成员 flag 值为 'x'。

#15：调用 investigate( ) 函数后，变量 stu 的成员 flag 值仍然为 'x'，没有改变。

#19～#24：定义 investigate( ) 函数，形参为结构体类型 student。

#21：修改形参的成员 flag 值。

程序运行结果如下：

```
Student flag:x
Warning:Zhang3's score is 52.0 < 60
Student flag:x
```

2. 结构体指针变量作为函数参数

使用结构体指针变量作为函数参数进行传递，是“地址传递”方式，能够减少时间和空间的开销，而且形参指针变量对目标变量所做的修改会影响实参所指向的结构体变量。

【例 8-6】 程序 8-6：结构体指针变量作为函数参数示例，将一个学生成绩结构体数组中不及格学生的 flag 值改为 'F'，并计算平均成绩。

```
#01: //程序 8-6
#02: #include <stdio.h>
#03: #define LEN 5
#04: struct student{
#05:     int id;
#06:     char* name;
#07:     float score;
```

```
#08:    char flag;
#09: };
#10: float investigate(struct student*,int);
#11: int main() {
#12:    float average;
#13:    struct student *pstu,stu[LEN]={{1,"Zhang3",85,'x'},
#14:                                   {2,"Li4",98,'x'},
#15:                                   {3,"Wang5",56,'x'},
#16:                                   {4,"Zhao6",78,'x'},
#17:                                   {5,"Sun7",46,'x'}};
#18:
#19:    average=investigate(stu,LEN);
#20:    printf("Student average score:%.1f\n",average);
#21:    for(pstu=stu;pstu<stu+LEN;pstu++){
#22:       if (pstu->score<60){
#23:          printf("Student id %d's flag:%c\n",pstu->id,pstu->flag);
#24:          printf("Warning:%s's score is %.1f < 60\n",pstu->name,pstu->score);
#25:       }
#26:    }
#27:    return 0;
#28: }
#29:
#30: float investigate(struct student *ps,int len){
#31:    int i;
#32:    float aver=0;
#33:
#34:    for(i=0;i<len;i++,ps++){
#35:       aver+=ps->score;
#36:       if (ps->score<60)
#37:          ps->flag='F';
#38:    }
#39:    aver/=len;
#40:    return aver;
#41: }
```

程序解释：

#19：调用 investigate( ) 函数，将数组首元素地址和数组长度传递给形参。

#21～#26：根据元素成员 score 值，输出 flag 值。flag 的值发生了改变。

#30：定义 investigate( ) 函数，形参为结构体类型指针和整型。

#34～#39：计算数组元素的 score 成员的平均值，并根据 score 值修改 flag 值。

程序运行结果如下：

```
Student average score:72.6
Student id 3's flag:F
Warning:Wang5's score is 56.0 < 60
Student id 5's flag:F
Warning:Sun7's score is 46.0 < 60
```

8.2 联 合 体

联合体（union）可以使用同一段内存单元存放不同类型的变量，这些变量的起始地址相同。不能把多个成员变量同时存入一个联合体变量中，而只能每次赋值一个成员变量，后赋值的成员变量值会覆盖以前的成员变量值。

联合体与结构体不同，结构体中各成员有独立的内存空间，一个结构体变量的长度是各成员长度之和；而联合体中各成员“共享”一段内存空间，一个联合体变量的长度是各成员长度中的最大值。

与结构体类似，必须先定义联合体类型，然后才能定义该类型的联合体变量。定义联合体类型的一般形式为：

```
union 联合体名{
    成员列表
};
```

“成员列表”由若干成员组成，每个成员都需要说明类型，其一般形式为：“类型标识符 成员名;”。例如，下列程序块定义了一个 aircraft 类型的联合体，含有三个成员，同时只有一个成员有效。

```
union aircraft{
    short passenger;
    float cargo_capacity;
    long long bomb_load;
};
```

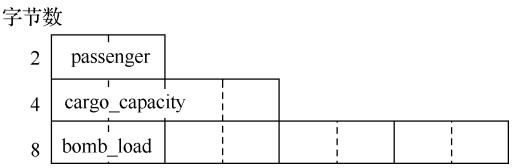


图 8-3 联合体存储结构

该联合体 aircraft 具有图 8-3 所示的存储结构。

联合体类型被定义后，可以定义联合体变量，被定义为 aircraft 类型的变量可以存放 short 型变量 passenger 或 float 型变量 cargo_capacity 或 long long 型变量 bomb_load。

定义联合体变量也有三种方式：①先定义联合体类型，再定义联合体变量；②定义联合体类型同时定义联合体变量；③不定义联合体类型，直接定义联合体变量。具体定义方法与结构体变量类似，在此不再赘述。例如，下列程序块先定义了一个 untype 类型的联合体，然后定义一个联合体变量 un1。

```
union untype{
    char ch;
    short s;
    int i;
};
union untype un1;
```

定义了联合体变量后可以在程序中引用。不能引用联合体变量，只能引用联合体变量中的成员，引用方法为“联合体变量名.成员名”，如“un1.ch”。

对联合体变量初始化时，初始化列表中只能有一个常量，如“union untype un1={‘x’};”，因为所有成员公用同一段存储空间，同一时刻只能存放其中一个成员。

对联合体变量赋值时，起作用的是最后一次被赋值的成员，以前存储单元中的内容被取代。例如，下列程序块对联合体变量 un1 赋值三次后，存储单元中只有“0xff”的值。

```
un1.ch='x';
un1.s=100;
un1.i=0xff;
```

联合体变量的地址和它的各成员变量的地址相同，如“&un1”、“&un1.ch”、“&un1.s”和“&un1.i”的值相同。

不能使用联合体变量名直接赋值或获取一个值，但是同类型的联合体变量可以相互赋值。

【例 8-7】 程序 8-7：联合体程序示例，用同一个表格表示教师与学生的信息，教师数据有编号、姓名、办公室三项，学生数据有编号、姓名、班级三项。

```
#01: //程序 8-7
#02: #include <stdio.h>
#03: #define LEN 3
#04: struct body{
#05:     int id;
#06:     char name[32];
#07:     char category;
#08:     union {
#09:         int class;
#10:         char office[32];
#11:     }addr;
#12: };
#13: int main(){
#14:     struct body person[LEN];
#15:     int i;
#16:
#17:     for(i=0;i<LEN;i++){
#18:         printf("Input id,name,category[s/t]:");
#19:         fflush(stdin);
#20:         scanf("%d %s %c",
                    &person[i].id,person[i].name,&person[i].category);
#21:         if (person[i].category=='s'){
#22:             printf("Input student class number:");
#23:             fflush(stdin);
#24:             scanf("%d",&person[i].addr.class);
#25:         }else if (person[i].category=='t'){
#26:             printf("Input teacher office add:");
#27:             fflush(stdin);
#28:             gets(person[i].addr.office);
#29:         }else{
#30:             printf("Input category error!\n");
#31:             return -1;
#32:         }
#33:     }
#34:
```

```
#35:    printf("ID\tname\tcategory\tclass/office\n");
#36:    for(i=0;i<LEN;i++){
#37:        if (person[i].category=='s')
#38:            printf("%d\t%s\tstudent\t\t%d\n",
                        person[i].id,person[i].name,person[i].addr.class);
#39:        else
#40:            printf("%d\t%s\tteacher\t%s\n",
                        person[i].id,person[i].name,person[i].addr.office);
#41:    }
#42:
#43:    return 0;
#44: }
```

程序解释：

#08～#11：定义一个无名的联合体类型，联合体变量为 addr。

#21～#32：根据变量 category 的值，决定是教师（'t'）还是学生（'s'）对联合体变量 addr 中 office 成员或 class 成员赋值。

#38：引用联合体变量 addr 的 class 成员。

#40：引用联合体变量 addr 的 office 成员。

程序运行结果如下：

```
Input id,name,category[s/t]:1 zhang3 s
Input student class number:11
Input id,name,category[s/t]:2 li4 t
Input teacher office add:room 601,building 2
Input id,name,category[s/t]:3 wang5 s
Input student class number:12
ID      name    category        class/office
1       zhang3  student         11
2       li4     teacher room 601,building 2
3       wang5   student         12
```

8.3 枚 举 类 型

枚举类型是一种基本数据类型，而不是一种构造类型。当一个变量的值被限定在一个有限范围内时，可以使用枚举类型，在定义中列举出所有值，被定义为该枚举类型的变量，取值不能超出定义的范围。例如，一周有 7 天，可以定义枚举类型，只有 0 到 6 之间的值，然后定义的变量只能取 0 到 6 之间的某个值。

定义枚举类型的一般形式为：“enum 枚举类型名{枚举值列表};”，其中“枚举值列表”列出所有的可用值，用逗号“,”隔开，称为枚举元素或枚举常量，枚举元素的值由用户根据自己的要求制定。

例如，“enum week{sunday, monday, tuesday, wednesday, thursday, friday, saturday};”定义了一个枚举类型 week，共有 7 个值，定义为 week 类型的变量的取值只能是其中一个。

定义了枚举类型后，可以定义该类型的枚举变量，定义枚举变量有三种方式：①先定义枚举类型，然后定义枚举变量，如“enum week workday, weekend;”；②定义枚举类型的同时定义枚举变量，如“enum

week{sunday, monday, tuesday, wednesday, thursday, friday, saturday}workday, weekend;”；③不定义有名字的枚举类型而直接定义枚举变量，如“enum{sunday, monday, tuesday, wednesday, thursday, friday, saturday}workday, weekend;”。

对于枚举类型和枚举变量，使用时应注意：

（1）对于枚举元素，按常量处理，不能对它们赋值，如“sunday=0”是非法的；

（2）枚举元素具有默认数值，从 0 开始依次为：0，1，2，…，如果有语句“workday=tuesday;”，相当于“workday=2”；也可以在定义时另行指定枚举元素的值，如“enum weekday{sunday=7, monday=1, tuesday, wednesday, thursday, friday, saturday};”；

（3）枚举元素可以进行关系运算，按照它们表示的数值进行比较，如“workday > sunday”；

（4）枚举变量输出时，按照所对应的枚举元素的整数值输出；

（5）整型数值不能直接赋值给枚举变量，如需要将整型数值赋值给枚举变量，应进行强制类型转换，其意义是将对应顺序号的枚举元素赋值给枚举变量。

【例 8-8】 程序 8-8：枚举变量示例，体育比赛的结果有 4 种可能：胜（win）、负（lose）、平局（tie）、比赛取消（cancel），输出比赛结果。

```
#01: //程序 8-8
#02: #include <stdio.h>
#03: enum game_result {win,lose,tie,cancel};
#04: int main(){
#05:     enum game_result result;
#06:     enum game_result omit=cancel;
#07:     int i;
#08:
#09:     for (i=win; i<=cancel;i++){
#10:         result = (enum game_result)i;
#11:         if (result == omit)
#12:           printf("Result:%d. The game was cancelled\n",result);
#13:         else{
#14:           printf("Result:%d. The game was played,",result);
#15:           if (result==win)
#16:                printf("we won!\n");
#17:           if (result==lose)
#18:                printf("we lose!\n");
#19:           if (result==tie)
#20:                printf("we tie!\n");
#21:         }
#22:     }
#23:     return 0;
#24: }
```

程序解释：

#03：定义枚举类型 game_result，只有 4 种取值：win、lose、tie 和 cancel。

#05, #06：定义两个枚举变量。

#09：枚举元素 win、cancel 如同整型数值使用。

#10：整型数值通过强制类型转换赋值给枚举变量。

#12, #14：输出枚举变量的值，为整型值。

#15～#20：对枚举变量进行关系运算。

程序运行结果如下：

```
Result:0. The game was played,we won!
Result:1. The game was played,we lose!
Result:2. The game was played,we tie!
Result:3. The game was cancelled
```

8.4 位运算符与位段

8.4.1 位运算符

前面的运算都是以字节作为最基本单位的，C 语言提供了位运算功能，能够在位（bit）一级进行运算或处理。C 语言提供了 6 种位运算符，如表 8-1 所示。

表 8-1 位 运 算 符

位 运 算 符	意 义	示 例
&	按位与	a&b
|	按位或	a|b
^	按位异或	a^b
～	求反	～a
<<	左移	a<<2
>>	右移	a>>2

1. 按位与运算符

按位与运算符“&”将参与运算的两个数各自对应的二进制数（补码形式）的对应位相与。“与”操作规则为：只有对应的两个位均为 1 时，结果位才为 1，否则为 0。

例如，99&199=67，计算过程如下：

	二进制		十进制
	01100011		99
&	11000111	&	199
	01000011		67

按位与运算通常用来对某些位清 0 或保留原值。例如，使用“a&0xFF”，保留变量 a 的最低一字节（低 8 位）。

2. 按位或运算符

按位或运算符“|”将参与运算的两个数各自对应的二进制数（补码形式）的对应位相或。“或”操作规则为：只要对应的两个位中有一个为 1 时，结果位就为 1。

例如，99|199=231，计算过程如下：

	二进制		十进制
	01100011		99
|	11000111	|	199
	11100111		231

3. 按位异或运算符

按位异或运算符“^”将参与运算的两个数各自对应的二进制数（补码形式）的对应位相异或。“异或”操作规则为：对应的两个位不同时为 1，相同时为 0。

例如，99^199=164，计算过程如下：

二进制		十进制	
01100011		99	
^ 11000111		^ 199	
10100100		164	

异或后的结果，再与之前的一个数值进行异或，可以得到另外一个数值。即如果“a^b=c”，则有“c^b=a”。

4．求反运算符

求反运算符“～”为单目运算符，具有右结合性。将参与运算的数对应的二进制数（补码形式）的对应位取反。

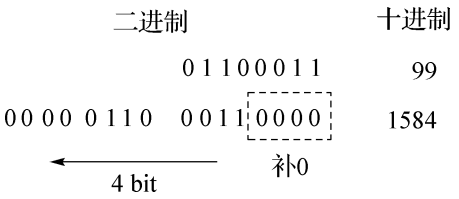
例如，～99= –100，计算过程（按单字节形式）如下：

二进制		十进制	
～ 01100011		～ 99	
10011100		–100	

5．左移运算符

左移运算符“<<”将“<<”左边的运算数对应的二进制数（补码形式）的各位向左移若干位，高位丢弃，低位补 0，移动位数由“<<”右边的数指定。

例如，99<<4=1584，计算过程（按 2 字节形式）如下：

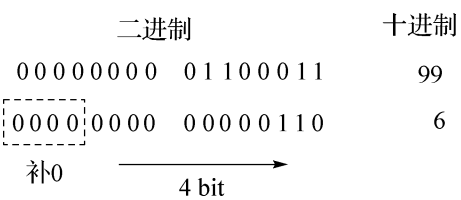


左移运算符“<<”相当于将数值乘以 2 的若干次幂，如“99<<4”等价于“99×2^4”。

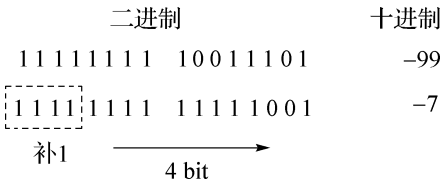
6．右移运算符

右移运算符“>>”将“>>”右边的运算数对应的二进制数（补码形式）的各位向右移若干位，移动位数由“>>”右边的数指定。对于无符号数，右移时高位补 0，低位丢弃。对于有符号数，当为正数时，高位补 0，低位丢弃；当为负数时，高位补 1，低位丢弃。

例如，99>>4=6，计算过程（按 2 字节形式）如下：



–99>>4 = –7，计算过程（按 2 字节形式）如下：



【例 8-9】 程序 8-9：位运算符示例。

```
#01: //程序 8-9
#02: #include <stdio.h>
#03: int main(){
#04:    unsigned int x;
#05:
#06:    x=64;
#07:    printf("x&(x-1)=%#x\n",x&(x-1));
#08:    x=63;
#09:    printf("x&(x+1)=%#x\n",x&(x+1));
#10:    x=0x58;
#11:    printf("x&(-x)=%#x\n",x&(-x));
#12:    x=0xA7;
#13:    printf("-x&(x+1)=%#x\n",-x&(x+1));
#14:    x=0x58;
#15:    printf("x|(x-1)=%#x\n",x|(x-1));
#16:    return 0;
#17: }
```

程序解释：

#07：“x&(x–1)”可以用来检查一个无符号整数是否为 2 的幂，结果为零时为真。

#09：“x&(x+1)”可以用来检查一个无符号整数是否为 2^n–1 的形式，结果为零时为真。

#11：“x&(–x)”可以用来分离最右侧的为“1”的比特位，“0101 1000”变为“0000 1000”，即最右侧的“1”位出现在第 3 位（从 0 开始）。

#13：“–x&(x+1)”可以用来分离最右侧的为“0”的比特位，“1010 0111”变为“0000 1000”，即最右侧的“0”位出现在第 3 位（从 0 开始）。

#15：“x|(x–1)”可以用来将最右侧的“1”比特位向右侧扩展，“0101 1000”变为“0101 1111”，即最右侧第 3 位的“1”位扩展到右侧第 0、1 和 2 位。

程序运行结果如下：

```
x&(x-1)=0
x&(x+1)=0
x&(-x)=0x8
-x&(x+1)=0x8
x|(x-1)=0x5f
```

8.4.2 位段

位段也称位域，将一个字节中的 8 个二进制位划分为几个不同的区域，指定每个区域的位数，用来存放不同的变量值。这样可以节省存储空间，并且处理较简便。例如，一个标志变量，只有 0 和 1 两个值，不需要占用一字节，使用一个比特位即可，这样一字节就可以存放 8 个这样的标志变量。

位段划分的每个区域都有一个“位段名”，允许在程序中按位段名进行引用，“位段名”称为“位段成员”。

位段类型的定义与结构体类型的定义类似，一般形式为：

```
Struct 位域结构体名{
    位段成员列表
};
```

其中“位段成员列表”的一般形式为：“类型说明符 位域名:位域长度;”。例如，下列程序块定义了一个 bs 类型的位段，共有三个成员，其中变量 a 占三个比特位、变量 b 占两个比特位、变量 c 占一个比特位。

```
struct bs{
    char a:3;
    char b:2;
    char c:1;
};
```

该位段 bs 具有图 8-4 所示的存储结构，可以看出，位段从一字节的低字节开始存放位段成员。

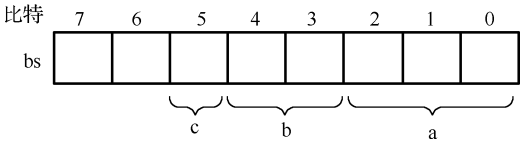


图 8-4　位段存储结构 1

位段可以看作是一种特殊的结构体，仅在于位段成员变量所占内存空间的大小不同。因此，定义位段变量与定义结构体变量的方式完全相同，例如，下列程序块定义了一个 bs 类型的位段变量 bs_val。

```
struct bs{
    char a:3;
    char b:2;
    char c:1;
};
struct bs bs_val;
```

位段成员的长度为任意值，只要不超过定义该位段成员的类型长度即可。如果一字节剩余空间不够存放另一位段成员时，则该位段成员将占用该字节剩余的空间，并从下一字节单元继续存放该位段成员剩余部分。因此位段成员可以不存储在同一字节中，能够跨越两个甚至多字节。例如，下列程序块中，位段变量 bs_val 的位段成员 a 和 b 一共占 7 个比特位，该字节只剩余一个比特位，而位段成员 c 需要两个比特位，因此位段成员 c 从将占用第一字节的一个比特位，以及下一字节的一个比特位。注意位段低字节开始存放位段成员。

```
struct bs{
    unsigned short a:4;
    unsigned short b:3;
    unsigned short c:2;
};struct bs bs_val={10,6,2};
```

该位段 bs 的位段变量 bs_val 具有图 8-5(a)所示的存储结构。

位段成员可以没有名字，此时用来作填充或调整后续位段成员的位置。无名位段成员不能被引用。例如，下列程序块中，位段变量 bs_val 中有一个无名位段成员，占据三个比特位，无须对该位段成员进行初始化。该位段 bs 的位段变量 bs_val 具有图 8-5(b)所示的存储结构。

```
struct bs{
    unsigned short a:4;
    unsigned short b:3;
    unsigned short  :3;
    unsigned short c:2;
};struct bs bs_val={10,6,2};
```

如果无名位段成员的长度指定为零，则后续位段成员将从一个新的数据类型存储单元开始。例如，下列程序块中，位段变量 bs_val 中有一个长度为 0 的无名位段成员，则位段成员 c 从一个新的 unsigned short 类型的存储单元开始，该位段 bs 的位段变量 bs_val 具有如图 8-5(c)所示的存储结构。

```
struct bs{
    unsigned short a:4;
    unsigned short b:3;
    unsigned short  :0;
    unsigned short c:2;
};struct bs bs_val={10,6,2};
```

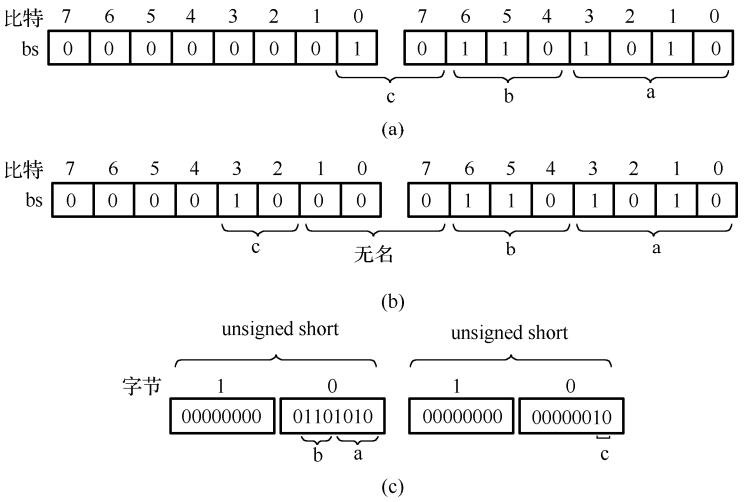


图 8-5　位段存储结构 2

位段的使用和结构体相同，引用方式为“位段变量名.位段成员名”，也可以使用指针指向位段变量。

【例 8-10】 程序 8-10：位段示例。

```
#01: //程序 8-10
#02: #include <stdio.h>
#03: struct fbit{
#04:     unsigned int fraction:23;//fraction
```

```
#05:    unsigned int exponent :8; //exponent
#06:    unsigned int sign:1;//sign
#07: };
#08:
#09: int main(){
#10:    float f = 8.5;
#11:    struct fbit fval;
#12:
#13:    fval = *(struct fbit *)&f;
#14:    printf("fraction:%x\n",fval.fraction);
#15:    printf("exponent:%d\n",fval.exponent);
#16:    printf("sign:%d\n",fval.sign);
#17:    return 0;
#18: }
```

程序解释：

根据 IEEE 754 标准，float 类型变量在内存中符号部分占 1 位，指数部分占 8 位，小数部分占 23 位。其中指数部分将阶码+3。该程序读取 float 类型变量各部分的数值。

#03～#07：定义一个 float 类型变量的位段，共占 4 字节，字段成员从低字节的低位开始分配比特位。

#13：fval 的内容为 float 类型值 8.5。$(8.5)_{20}=1.0625\times2^3$。

#14：输出小数部分，二进制比特位为“000 1000 0000 0000 0000”，因此小数部分为二进制“0.0001”，即十进制 0.0625。

#15：输出指数部分，130=127+3，因此阶码为 3。

#16：输出符号部分，0 表示正数。

程序运行结果如下：

```
fraction:80000
exponent:130
sign:0
```

8.5　类型定义符 typedef

类型定义符 typedef 允许用户自己定义类型说明符，为数据类型取“别名”。使用 typedef 进行类型定义的一般形式为：“typedef 原类型名 新类型名;”，其中“新类型名”标识符的首字母一般用大写表示。例如，下列程序块代码定义了新类型 Integer，可用 Integer 代替 int 做整型变量的类型说明，“Integer a,b;”等价于“int a,b;”。

```
typedef int Integer
Integer a,b;
```

使用 typedef 可以定义数组、指针、结构等类型，例如，“typedef char String[32];”表示 String 是一个长度为 32 的字符数组类型，“String name;”等价于“char name[32]”。下列程序块代码定义了新类型 Date，表示结构体类型的日期，可用 Date 定义结构体变量：“Date birthday,holiday;”。

```
typedef struct {
    int year;
    int month;
```

```
    int day;
} Date;
```

typedef 只是对已经存在的类型指定一个新的类型名，不能创造新的类型。另外，对 typedef 的处理是在编译阶段进行的。

8.6 程 序 示 例

【例 8-11】 程序 8-11：每个学生的信息有学号、姓名、*N* 门课成绩。找出其中平均成绩最高的学生，并输出该学生的信息。

```
#01: //程序 8-11
#02: #include <stdio.h>
#03: #define LEN 5
#04: #define N 3
#05: struct student{
#06:    int id;
#07:    char name[32];
#08:    float score[N];
#09:    float average;
#10: };
#11: void input(struct student*,int);
#12: int find_max(struct student*,int);
#13: void output(struct student*,int);
#14:
#15: int main() {
#16:    struct student stu[LEN];
#17:    int index;
#18:
#19:    input(stu,LEN);
#20:    index=find_max(stu,LEN);
#21:    output(stu,index);
#22:    return 0;
#23: }
#24:
#25: void input(struct student* pstu,int num){
#26:    struct student *ps;
#27:    int i,aver;
#28:    for(ps=pstu;ps<pstu+num;ps++){
#29:       printf("Input %d student's id,name:",ps-pstu);
#30:       scanf("%d %s",&ps->id,ps->name);
#31:       fflush(stdin);
#32:
#33:       printf("Input %d student's %d course scores:",ps-pstu,N);
#34:       aver=0;
#35:       for(i=0;i<N;i++){
#36:          scanf("%f",&ps->score[i]);
```

```
#37:            aver+=ps->score[i];
#38:        }
#39:        ps->average=aver/3.0;
#40:    }
#41: }
#42:
#43: int find_max(struct student* pstu,int num){
#44:    struct student *ps;
#45:    int i,pos;
#46:
#47:    pos=0;
#48:    for(i=1;i<num;i++)
#49:        if ((pstu+i)->average > (pstu+pos)->average)
#50:            pos=i;
#51:
#52:    return pos;
#53: }
#54:
#55:
#56: void output(struct student* pstu,int index){
#57:    int i;
#58:
#59:    printf("Higest average student info:\n");
#60:    printf("\tID:%d,name:%s\n",(pstu+index)->id,(pstu+index)->name);
#61:    printf("\t3 course scores:");
#62:    for(i=0;i<N;i++)
#63:        printf("%.1f\t",(pstu+index)->score[i]);
#64:    printf("\n");
#65:    printf("\taverage:%.1f\n",(pstu+index)->average);
#66: }
```

程序解释：

#03, #04：定义学生数为 LEN，课程数为 N。

#19：调用 input( )函数输入 LEN 个学生的信息。

#20：调用 find_max( )函数查找平均成绩最高的学生，返回该学生在数组中的下标位置。

#21：调用 output( )函数，输出下标位置为 index 的学生信息。

#25～#41：定义 input( )函数，输入 N 门成绩后，计算平均值并保存。

#43～#53：定义 find_max( )函数，查找平均成绩最高的学生，位置信息保存在 pos 中并返回。

#56～#66：定义 output( )函数，输出指定下标位置的学生信息。

程序运行结果如下：

```
Input 0 student's id,name:15001 zhang3
Input 0 student's 3 course scores:95 85 82
Input 1 student's id,name:15002 li4
Input 1 student's 3 course scores:90 99 88
Input 2 student's id,name:15003 wang5
Input 2 student's 3 course scores:92 93 95
```

```
Input 3 student's id,name:15004 zhao6
Input 3 student's 3 course scores:98 95 99
Input 4 student's id,name:15005 qian7
Input 4 student's 3 course scores:80 85 90
Higest average student info:
       ID:15004,name:zhao6
       3 course scores:98.0    95.0    99.0
       average:97.3
```

上机实验：结构体程序设计应用

本次实验掌握 C 语言程序的结构体的定义与使用，熟练掌握结构体数组、结构体指针与结构体函数的用法。

有 *N* 个学生，有学号、姓名、成绩等信息，按照成绩从低到高的顺序排序并输出。

程序示例：

```
#include <stdio.h>
#define LEN 5
struct student{
    int id;
    char name[32];
    float score;
};
void input(struct student*,int);
void sort(struct student*,int);
void output(struct student*,int);

int main() {
   struct student stu[LEN];

   input(stu,LEN);
   sort(stu,LEN);
   printf("Sorted score:\n");
   output(stu,LEN);
   return 0;
}

void input(struct student* pstu,int num){
   struct student *ps;
   for(ps=pstu;ps<pstu+num;ps++){
      printf("Input %d student's id,name,socre:",ps-pstu);
      scanf("%d %s %f",&ps->id,ps->name,&ps->score);
      fflush(stdin);
   }
}
```

```
void sort(struct student* pstu,int num){
    struct student temp;
    int i,j,min;

    for(i=0;i<num-1;i++){
        min=i;
        for(j=i+1;j<num;j++)
            if ((pstu+min)->score>(pstu+j)->score)
                min=j;

        if (min!=i){
            temp=*(pstu+min);
            *(pstu+min)=*(pstu+i);
            *(pstu+i)=temp;
        }
    }

}

void output(struct student* pstu,int num){
    struct student *ps;
    for(ps=pstu;ps<pstu+num;ps++)
        printf("ID:%d,\tname:%s,\tscore:%.1f\n",ps->id,ps->name,ps->score);
}
```

习　题

1．定义一个表示坐标点（横坐标、纵坐标）的结构体类型，输入两点，计算两点之间的距离。

2．定义一个表示时间（小数、分钟、秒）的结构体类型，输入一个时间变量和一个整型变量 n，计算并输出该时间点之后 n 秒的时间值。

3．定义一个表示日期（年、月、日）的结构体类型，输入一个日期，计算该日期在该年中是第几天，注意考虑闰年的情况。

4．定义一个表示复数（实部、虚部）的结构体类型，并求两个复数相加的结果。

5．定义一个表示学生信息（学号、姓名、性别、成绩）的结构体类型：

（1）通过指向结构体变量的指针变量来输入和输出其中的信息；

（2）有 *N* 个学生的信息，放在结构体数组中，输入和输出全部学生的信息。

6．有 *N* 个候选人，每个选民只能投票选一个人，编写函数统计选票，输入每张选票上的候选人的姓名，输出每个候选人的得票结果。

7．编写一个函数 getbits(m, n)从一个 32 位的单元中取出以 m 开始至 n 结束的某几位，起始位和结束位都从右向左计算。

第 9 章　编译预处理

编译预处理是指在编译器进行第一遍扫描（词法扫描和语法分析）之前所做的工作，由预处理程序负责完成，编译时先调用预处理程序对源文件中的预处理指令进行处理，处理完毕后对源程序进行编译。以“#”符号开始的指令称为编译预处理指令，一般放在源文件的开始部分，例如，“#include”为文件包含指令。

C 语言的编译预处理指令有文件包含、宏定义、条件编译等几类。

9.1　文 件 包 含

文件包含指令可以将其他源文件内容“引入”到当前源文件中，被包含的文件是源文件的一部分，编译时以包含处理后的文件为编译单位。文件包含指令在模块化程序设计中很有用，一个较大的程序可以分成多个模块，由多名程序员分别实现，公用的符号常量、全局变量、公用的函数等内容可单独组成一个文件，在其他文件开始位置用文件包含指令包含该文件，避免了在每个文件开始位置都要书写这些内容，提高了效率。

文件包含指令的一般形式为：“#include "文件名"”或 “#include <文件名>”，被包含的“文件名”必须用双引号（" "）或一对尖括号（< >）括起来，通过该指令，用指定的文件取代该行，从而把指定的文件和当前的源文件连成一个源文件。

被包含的“文件名”用双引号和用尖括号括起来有如下区别：使用双引号（" "）表示首先在当前的源文件目录中查找该文件，若未找到，就到预定义的默认目录下（由操作系统给定）去查找；使用尖括号（< >）表示在预定义的默认目录下去查找该文件，而不在源文件目录查找。通常，“#include <file>”指令一般用来包含标准头文件（如 stdio.h、stdlib.h 等），这些头文件极少被修改，并且它们总是存放在编译程序能够找到的目录下。“#include "file"”指令一般用来包含非标准头文件，因为这些头文件一般存放在当前目录下。

文件包含允许嵌套包含，即在一个被包含的文件中又可以包含另外一个文件。例如，下列程序块中源文件 a.c 包含了 b.c，源文件 b.c 又包含了 c.c，则源文件最终的内容同时包含了 b.c 和 c.c 的内容。

```
a.c:
#include “b.c”

b.c:
#include “c.c”
```

一条#include 指令只能指定一个被包含的文件，如果有多个文件要包含，则必须用多个#include 指令。例如，下列程序块包含了三个文件：stdio.h、math.h 和 string.h。

```
#include <stdio.h>
#include <math.h>
#include <string.h>
```

#include 指令包含文件时，也可以指定目录，如“#include "test/user.c"”表示将 test 目录（test 在当前目录下）下的 user.c 文件包含进来。

【例 9-1】 程序 9-1：文件包含示例。

```
#01: //程序 9-1
#02: #include <stdio.h>
#03: #include "prime.h"
#04: int main() {
#05:     int i;
#06:
#07:     for(i=0;i<100;i++)
#08:         if (judge_prime(i))
#09:             printf("%d\t",i);
#10:
#11:     return 0;
#12: }
prime.h:
#01: #include <math.h>
#02: int judge_prime(int num){
#03:     int i,flag=1;
#04:     for(i=2;i<=sqrt(num);i++)
#05:         if (num%i==0){
#06:             flag=0;
#07:             break;
#08:         }
#09:
#10:     return  (flag);
#11: }
```

程序解释：

在程序 9-1 源文件中：

#03：使用文件包含指令将 prime.h 的内容包括进来。

#08：可以使用 prime.h 文件中定义的函数。

prime.h 文件称为头文件，与程序9-1 源文件放在同一个目录下。

在 prime.h 文件中：

#02：定义了一个判断一个数 num 是否为素数的函数 judge_prime( )。

程序运行结果如下：

```
1   2    3   5   7  11  13  17  19  23
29  31  37  41  43  47  53  59  61  67
71  73  79  83  89  97
```

9.2 宏 定 义

宏定义是指采用“#define”指令用一个标识符表示一个字符串，该标识符称为“宏名”，字符串成为“宏体”。通常“宏名”用大写字母表示，以与变量进行区别。在编译预处理阶段，预处理程序将源程序中所有的宏名替换成宏定义中的字符串，称为宏替换或宏展开。

使用“#define”指令进行宏定义后，可以使用“#undef”指令取消以前进行的宏定义，即“不定义”该宏名，一般形式为：“#undef 宏名”。

根据宏替换时是否传递参数，宏定义可分为无参数宏定义和带参数宏定义两种。

9.2.1 无参数宏定义

无参数宏定义的宏名后面不带参数，定义无参数宏的一般形式为：“#define 宏名标识符 宏体字符串”，其中“宏体字符串”可以是常数、表达式、格式化字符串等。例如，前面章节用来定义常量的方法“#define LEN 5”就是一个无参数宏定义。

宏定义的作用域从宏定义指令开始到源程序结束，或者遇到“#undef”指令取消该宏名为止。

通常对程序中反复使用的表达式进行无参数宏定义，如“#define EXPR(x*x+x+1)”定义了一个宏EXPR，代表字符串“(x*x+x+1)”。编写程序时，可以使用宏名 EXPR 代替字符串“(x*x+x+1)”；编译时，预处理程序使用字符串“(x*x+x+1)”去替换所有的宏名 EXPR，然后再进行编译。

宏定义不是一条语句，在末尾不能加分号“;”，如果有分号，则分号也是“宏体”的一部分，宏展开时连分号也一起替换。

宏定义是用“宏名”表示一个宏体字符串，宏展开时又以该字符串替换“宏名”，仅做简单的替换，宏体字符串可以含任何字符，预处理程序对宏体字符串不做检查。如果有错误，只能在宏展开后编译源程序时发现。

【例 9-2】 程序 9-2：无参数宏定义示例。

```
#01: //程序 9-2
#02: #include <stdio.h>
#03: #define EXPR (x*x+x+1)
#04: int main(){
#05:     float result,x;
#06:
#07:     printf("Input a val:");
#08:     scanf("%f",&x);
#09:     result=EXPR+x*EXPR;
#10:     printf("result=%.3f\n",result);
#11:     return 0;
#12: }
```

程序解释：

#03：进行宏定义，宏 EXPR 代表字符串“(x*x+x+1)”。

#09：进行宏 EXPR 调用，在编译预处理时进行宏展开，将 EXPR 替换成“(x*x+x+1)”，即该行等价于：“result=(x*x+x+1)+x*(x*x+x+1);”。

程序运行结果如下：

```
Input a val:1.5
result=11.875
```

对于无参数宏定义，应注意以下几点。

（1）宏定义中的“宏体”两边通常需要括号“( )”，如果省略，可能会发生错误。例如，例 9-2 中，如果“#define EXPR x*x+x+1”，则宏展开时将得到：“result= x*x+x+1+x*x*x+x+1;”，结果与预期不同。

（2）对于源程序中用引号括起来的与宏名相同的字符串，预处理程序不对其进行宏展开。例如，下列程序块中语句“printf("EXPR=%.3f\n",result);”中的“EXPR”用引号括了起来，是一个字符串的一部分，不会做宏展开。

```
#define EXPR (x*x+x+1)
int main(){
```

```
    …
    printf("EXPR=%.3f\n",result);
    …
}
```

（3）宏定义允许嵌套定义，在宏定义的宏体字符串中可以使用已经定义过的宏名，在宏展开时由预处理程序自动进行层层替换，但是宏定义不能递归定义。例如，下列程序块中先定义宏 PI，然后使用宏 PI 定义了宏 AREA，又使用宏 AREA 和宏 PI 定义了宏 SURFACE，如果程序中有语句“printf("SURFACE=%f\n", SURFACE);”，则宏展开后为“printf("SURFACE=%f\n",2*3.14*r*r+2*3.14*r*h);”。但是宏 MAX 的嵌套定义是错误的。

```
#define PI 3.14
#define AREA PI*r*r
#define SURFACE 2*AREA+2*PI*r*h
#define MAX MAX+1 //ERROR
```

（4）可以使用宏定义表示数据类型，方便编写程序。例如，“#define INTEGER int”进行宏定义 INTEGER 后，可以使用语句“INTEGER a,b;”定义整型变量 a 和 b。又如宏定义：“#define STU struct student”，可以使用语句“STU stu1, stu[5],*pstu;”定义 student 结构体变量 stu1、结构体数组 stu[5]和结构体指针变量 pstu。

使用宏定义#define 和类型定义符 typedef 表示数据类型不同，宏定义#define 在预处理阶段完成，只是简单地进行字符串替换；类型定义符 typedef 在编译阶段进行处理，不是简单的替换，而是对类型说明符的重命名，被命名的标识符具有类型说明符的功能。例如，下列程序块中定义了一个宏 INT_PTR1 和新类型 Int_Ptr2，“INT_PTR1 a1,b1;”宏展开后为“int* a1,b1;”，即定义了一个整型指针变量 a1 和整型变量 b1；而 Int_Ptr2 是一个类型说明符，“Int_Ptr2 a2,b2;”等价于“int *a2,*b2;”

```
#define INT_PTR1 int*
typedef int* Int_Ptr2;
int main(){
    INT_PTR1 a1,b1;
    Int_Ptr2 a2,b2;
    …
}
```

9.2.2　带参数宏定义

宏定义时可以带参数，宏定义中的参数称为形式参数（形参），宏调用中的参数称为实际参数（实参），在预处理阶段进行宏展开时，要用实参去替换形参，其他字符保留不变。

带参数宏定义的一般形式为：“#define 宏名（形参列表）宏体字符串”，其中“宏名”和后面的“（形参列表）”之间不能有空格，宏体字符串中含有各个形参。程序中使用带参数宏的方式为：“宏名（实参列表）”。

例如，有宏定义“#define AREA(a,b) a*b”，当在程序中使用宏“area=AREA(3,2);”时，首先将实参 3、2 传递给形参 a、b（①）；然后将宏体字符串中的 a 替换为 3、b 替换为 2（②），这样宏展开成“area=3*2;”（③），如图 9-1 所示。

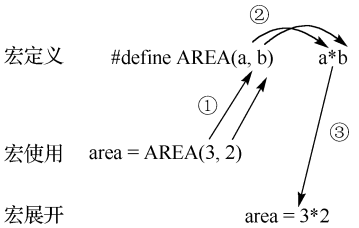


图 9-1　带参数宏展开

【例 9-3】 程序 9-3：带参数宏定义示例。

```
#01: //程序 9-3
#02: #include <stdio.h>
#03: #define MAX(a,b) (a)>(b)?(a):(b)
#04: int main(){
#05:     int x,y,max;
#06:
#07:     printf("Input two numbers:");
#08:     scanf("%d%d",&x,&y);
#09:     max=MAX(x, y);
#10:     printf("max(%d,%d)=%d\n",x+y,x-y,max);
#11:     return 0;
#12: }
```

程序解释：

#03：定义一个带参数的宏 MAX，形参为 a 和 b。

#09：使用该宏，实参为 x 和 y，宏展开为“(x)>(y)?(x):(y)”。

程序运行结果如下：

```
Input two numbers:1 2
max(1,2)=2
```

对于带参数宏定义中的形参，无须进行类型定义，不分配内存单元；而使用宏时的实参可以是常量、变量或表达式，宏展开时按照形参原样（对实参表达式不做计算求值）进行符号替换，不进行值传递。这与函数中的形参和实参概念不同，函数调用时要先对实参表达式求值，然后把值传递给形参。

【例 9-4】 程序 9-4：实参为表达式情况示例。

```
#01: //程序 9-4
#02: #include <stdio.h>
#03: #define SQUARE(a) (a)*(a)
#04: int main(){
#05:     int x,y,result;
#06:
#07:     printf("Input two numbers:");
#08:     scanf("%d%d",&x,&y);
#09:     result=SQUARE(x+y);
#10:     printf("SQUARE(%d)=%d\n",x+y,result);
#11:     return 0;
#12: }
```

程序解释：

#03：定义一个带参数的宏 SQUARE，形参为 a。

#09：使用该宏，实参为 x+y，用“x+y”替换 a，然后用“(a)*(a)”替换 SQUARE，最后宏展开为“(x+y)*(x+y)”。先不对实参求值，而是按照表达式原样形式进行宏展开，然后再求值。

程序运行结果如下：

```
Input two numbers:3 4
SQUARE(7)=49
```

宏定义时的宏体字符串中的形参通常用括号“( )”括起来以避免在宏展开时出错。例如，例 9-4 中的宏体字符串“(a)*(a)”中的形参 a 用括号括起来。如果去掉括号，改为宏定义：“#define SQUARE(a) a*a”，则对于语句“result=SQUARE(x+y);”，由于宏展开时只做符号替换而不做其他处理，因此宏展开后为“x+y*x+y”，运算结果为 19（即 3+4*3+4）。

除了在参数两边加括号外，通常整个宏体字符串也要加上括号，例如，例 9-4 中，如果使用宏的语句为“result=147/SQUARE(x+y);”，则宏展开后为“result=147/(x+y)*(x+y);”，由于运算符“/”和“*”优先级相同，先进行“/”运算然后进行“*”运算，计算结果为 147。如果宏定义为“#define SQUARE(a) ((a)*(a))”，则宏展开后为“result=147/((x+y)*(x+y));”，则运算结果为 3（即 147/(7*7)）。因此对于宏定义，不仅要在形参的两侧加括号，还要在宏体字符串外加括号。

宏定义中的宏体字符串可以对应多条语句，宏展开时，这些语句都被替换到源程序中。

【例 9-5】 程序 9-5：宏体字符串为多条语句。

```
#01: //程序 9-5
#02: #include <stdio.h>
#03: #define SWAP(a,b,t) {(t)=(a);(a)=(b);(b)=(t);}
#04: int main(){
#05:     int x,y,temp;
#06:
#07:     printf("Input two numbers:");
#08:     scanf("%d%d",&x,&y);
#09:     SWAP(x,y,temp)
#10:     printf("x=%d,y=%d\n",x,y);
#11:     return 0;
#12: }
```

程序解释：

#03：定义宏 SWAP，如果宏体为多条语句，通常用“{}”括起来。功能是交换形参 a 和 b 的值。

#09：使用宏 SWAP，宏展开后宏体字符串替换到该位置。

程序运行结果如下：

```
Input two numbers:1 2
x=2,y=1
```

使用宏定义和函数定义虽然在形式上相似，都能完成相同功能，但在本质上是完全不同的，程序中执行函数时，将实参的值传递给形参，调用函数，函数执行完后返回到主调函数中继续执行；而程序中执行宏时，仅仅将宏体字符串展开，替换程序中的宏名。当把同一个表达式用函数处理和用宏处理时，结果可能不同。

【例 9-6】 程序 9-6：宏定义和函数定义区别。

```
#01: //程序 9-6
#02: #include <stdio.h>
#03: #define SQUARE(a) ((a)*(a))
#04: int square(int a){
#05:     return (a*a);
#06: }
#07:
#08: int main(){
```

```
#09:    int i;
#10:
#11:    printf("Function:\n");
#12:    for(i=0;i<5;)
#13:        printf("%d\n",square(i++));
#14:
#15:    printf("Macro:\n");
#16:    for(i=0;i<5;)
#17:        printf("%d\n",SQUARE(i++));
#18:    return 0;
#19: }
```

程序解释：

#03：定义一个宏 SQUARE，求形参 a 的平方。

#04～#07：定义一个函数 square，返回形参 a 的平方。

#12～#13：循环 5 次求 i 的平方，并对 i 自增 1，输出结果与预期相同。

#16～#17：SQUARE(i++)被宏展开为“((i++)*(i++))”，因此计算完 i 的平方后，对 i 自增加 2 次，等价于“i*i; i+=2;”，因此仅输出 i 为偶数的平方值。

程序运行结果如下：

```
Function:
0
1
4
9
16
Macro:
0
4
16
```

9.3 条 件 编 译

条件编译是指使用若干编译指令选择性地编译源代码的不同部分，这样可以根据不同的条件编译不同的程序部分，产生不同的可执行程序，方便程序的调试和移植。

条件编译有两种形式：①使用#if 系列（#if、#else、#elif 和#endif）编译指令；②使用#ifdef 和#ifndef 编译指令。

9.3.1 #if 系列编译指令

#if 系列（#if、#else、#elif 和#endif）编译指令允许用户根据常数表达式的值有条件地编译部分代码，一般形式为：

```
#if 常量表达式 1
    程序段 1
#elif 常量表达式 2
    程序段 2
```

#else
　　程序段 3
#endif

其作用是：如果常量表达式 1 的值为真（非 0），则对程序段 1 进行编译，否则判断常量表达式 2 的值，如果为真，则对程序段 2 进行编译，否则对程序段 3 进行编译。#elif 编译指令部分可以出现多次，#else 指令部分可以省略。

例如，下列程序中，根据 LEN 不同的值，会选择不同的程序段编译，例如，假设 LEN 值为 50，则会选择#elif 和#else 之间的程序段编译，因而运行程序时会输出“Compiled this part if 49<LEN<99.”。

```
#include<stdio.h>
#define LEN 50
int main(){
   #if LEN>99
      printf("Compiled this part if LEN>99.\n");
   #elif LEN>49
      printf("Compiled this part if 49<LEN<99.\n");
   #else
      printf("Compiled this part if LEN<50.\n");
   #endif
   return 0;
}
```

条件编译功能也可以用选择结构程序实现，但是用选择结构对整个源代码进行编译时，生成的目标程序较长；而条件编译根据条件只编译其中一部分程序段，生成的目标程序较短。

9.3.2　#ifdef 和#ifndef 编译指令

#ifdef 编译指令的一般形式为：

#ifdef 宏名
　　程序段 1
#else
　　程序段 2
#endif

其作用是：如果宏名已经被 #define 命令定义过，则对程序段 1 进行编译；否则对程序段 2 进行编译。#else 指令部分可以省略。

#ifndef 编译指令与#ifdef 编译指令的功能相反，一般形式为：

#ifndef 宏名
　　程序段 1
#else
　　程序段 2
#endif

其作用是：如果宏名没有被#define 命令定义过，则对程序段 1 进行编译，否则对程序段 2 进行编译。#else 指令部分可以省略。

【例 9-7】 程序 9-7：条件编译指令示例。

```
#01: //程序 9-7
#02: #include<stdio.h>
#03: #define DEBUG
#04: #define RUN
#05: int main(){
#06:    int x,y;
#07:
#08:    printf("Input two numbers:");
#09:    scanf("%d%d",&x,&y);
#10:    #ifdef DEBUG
#11:       printf("x=%d,y=%d\n",x,y);
#12:    #endif
#13:
#14:    #ifndef RUN
#15:       printf("x+y=%d\n",x+y);
#16:    #endif
#17:    return 0;
#18: }
```

程序解释：

#03, #04：定义宏名 DEBUG 和 RUN。

#10～#12：如果定义了 DEBUG，则输出变量 x 和 y 的值。在程序调试时，可以使用该语句来输出其中变量的值。

#14～#16：如果没有定义 RUN，则输出 x+y 的值。

程序运行结果如下：

```
Input two numbers:1 2
x=1,y=2
```

除 #ifdef 编译指令外，defined 编译指令也可以确定是否定义了宏名，一般形式为“defined 宏名”，作用是：如果宏名已经定义过，则“defined 宏名”为真，否则为假；也可以在 defined 之前加上非运算符“!”来反转相应条件。

通常将 #if 编译指令与 defined 编译指令一起使用，例如，“#if defined DEBUG”与“#indef DEBUG”是等价的。

9.4 其他预处理指令

9.4.1 操作符 # 和

除了文件包含、宏定义、条件编译等几类预处理指令外，还有两个预处理操作符：操作符“#”和操作符“##”，通常用在#define 编译指令中。

（1）操作符“#”

操作符“#”称为字符串化操作符，一般形式为“#字符串”，作用是把其后的字符串变成用双引号包围的串。例如，下列程序段中，“STR(a long string)”将参数转换成一个字符串，语句“printf("%s\n", STR(a long string));”等价于“printf("%s\n", "a long string");”。

```
#define STR(x) #x
printf("%s\n", STR(a long string));
```

又如，如果在调试程序时用宏“PRINT_INT”来输出整型变量或表达式的值，可以使用“#”为每个输出值添加变量信息，定义宏 PRINT_INT：“#define PRINT_INT(x)；printf(#x " = %d\n",x)；”，则使用宏“PRINT_INT(a/b);”时，会宏展开为：“printf("a/b" " = %d\n",a/b);”，即“printf("a/b = %d\n",a/b);”。

（2）运算符“##”

运算符“##”称为连接操作符，一般形式为“标识符 1##标识符 2”，作用是把“标识符 1”和“标识符 2”连接成一个标识符“标识符 1 标识符 2”。例如，下列程序段中，宏使用时“TYPE1(int, d);”被展开为“int name_int_type”，而“TYPE2(int, d);”被展开为“int d_int_type”，注意两者的区别。

```
#define TYPE1(type,name)   type name_##type##_type
#define TYPE2(type,name)   type name##_##type##_type
TYPE1(int, d);
TYPE2(int, d);
```

利用“##”运算符可以动态地构造出变量或要调用的函数名称。

【例 9-8】 程序 9-8：运算符“##”示例。

```
#01: //程序 9-8
#02: #include<stdio.h>
#03: #define VAR(v,x) v##x
#04: #define FUN(x,a,b) fun_##x((a),(b))
#05: void fun_add(int x,int y){
#06:     printf ("%d+%d=%d\n",x,y,x+y);
#07: }
#08: void fun_sub(int x,int y){
#09:     printf ("%d-%d=%d\n",x,y,x-y);
#10: }
#11: int main(){
#12:     int a1=8,b1=9, a2=5,b2=6;
#13:     char ch;
#14:
#15:     printf("Input + or -:");
#16:     ch=getchar();
#17:     if (ch=='+')
#18:         FUN(add,VAR(a,1),VAR(b,1));
#19:     else
#20:         FUN(sub,VAR(a,2),VAR(b,2));
#21:     return 0;
#22: }
```

程序解释：

#03：定义宏 VAR，用于构造变量 vx。

#04：定义宏 FUN，用于构造函数 fun_x(a,b)。

#18：宏展开为“fun_add(a1,b1);”。

#19：宏展开为“fun_sub(a2,b2);”。

程序运行结果如下：

```
Input + or -:-
5-6=-1
```

9.4.2 预定义宏

C 语言预定义了一些宏，这些宏在调试程序时很有用，可以知道程序运行到了哪个文件的哪一行、哪个函数等。这些宏的意义如表 9-1 所示。

【例 9-9】 程序 9-9：预定义宏示例。

```
#01: //程序 9-9
#02: #include<stdio.h>
#03: void fun(){
#04:     printf("In fun: This line is No %d.\n", __LINE__);
#05:     printf("In fun: In function:%s.\n", __func__);
#06: }
#07: int main(){
#08:     printf("Compile file:%s.\n", __FILE__);
#09:     printf("Compile Date:%s.\n", __DATE__);
#10:     printf("Compile Time:%s.\n", __TIME__);
#11:     printf("This line is No %d.\n", __LINE__);
#12:     printf("In function:%s.\n", __func__);
#13:     fun();
#14:     return 0;
#15: }
```

程序解释：

#04, #11：输出当前代码所在的行号。

#05, #12：输出当前代码所在的函数名。

#08：输出当前的源文件名，包括路径。

#09, #10：输出编译时的日期和时间。

程序运行结果如下：

表 9-1 预 定 义 宏

预 定 义 宏	意　义
DATE	源文件编译日期，格式“Mmm dd yyyy”
TIME	源文件编译时间，格式“hh:mm:ss”
FILE	被编译的源文件名
LINE	当前正在编译的行在源文件中的行号
func	当前正在编译的行所在函数的名称

```
Compile file:D:\C Book\Code\9\9-9.c.
Compile Date:May 11 2015.
Compile Time:14:23:16.
This line is No 11.
In function:main.
In fun: This line is No 4.
In fun: In function:fun.
```

9.5 程 序 示 例

【例 9-10】 程序 9-10：文件包含和宏定义示例。

```
#01: #include <stdio.h>
#02: #include "powers.h"
```

```
#03: #define N 6
#04: int main(){
#05:    int i;
#06:    printf("number\texp2\texp3\texp4\n");
#07:    for(i=0;i<N;i++)
#08:       printf("%3d\t%3d\t\%3d\t%3d\t\n",i,square(i),cube(i),quarter(i));
#09:    return 0;
#10: }

powers.h 文件:
#11: #define square(x) ((x)*(x))
#12: #define cube(x) ((x)*(x)*(x))
#13: #define quarter(x) ((x)*(x)*(x)*(x))
```

程序解释：

#02：包含 powers.h 文件。

#03：宏定义 N。

#08：调用宏 square、cube 和 quarter，这些宏在 powers.h 文件中定义过。

#11～#13：powers.h 文件中定义宏 square、cube 和 quarter。

程序运行结果如下：

```
number   exp2    exp3    exp4
  0       0       0       0
  1       1           1   1
  2       4       8      16
  3       9      27      81
  4      16      64     256
  5      25     125     625
```

习　题

1．编写一个宏求圆面积，并使用该宏求圆环的面积。

2．编写一个宏求三个整数中的最大数，并使用该宏输入三个整数，输出最大值。

3．编写一个宏求两个数的最小值，并使用该宏求出三个数中的最小数。

第 10 章　文　　件

10.1　文件与文件指针

文件是存储在外部介质（如磁盘）中的一组数据的有序集合，在使用时会读取到内存中。这个数据集有一个名称，称为“文件名”，可以通过文件名对文件进行操作。

文件通常用来存放数据，在程序运行时，经常需要从磁盘中读取数据到计算机内存中，或者将程序运行的中间数据或最终结果写入到磁盘中存放起来，这类文件称为数据文件。

根据文件内容的编码方式，数据文件分为两类：文本文件和二进制文件。

（1）文本文件，也称 ASCII 文件。文件存储时，磁盘中每字节存放一个字符的 ASCII 代码。例如，一个 int 型数据 98765，每个字符占 1 字节，在磁盘中一共占 5 字节，内容为“00111001000111000011011100110110001101011”（每 8 比特对应一个 ASCII 字符），如图 10-1(a)所示。

（2）二进制文件，也称映像文件。数据在内存中按二进制的编码形式存储，保存在文件中时对内存数据不加转换地映像保存。例如，一个 int 型数据 98765，内存用 4 字节存储，保存在磁盘中一共占 4 字节，内容为“00000000000000001100000011100 1101”，如图 10-1 (b)所示。

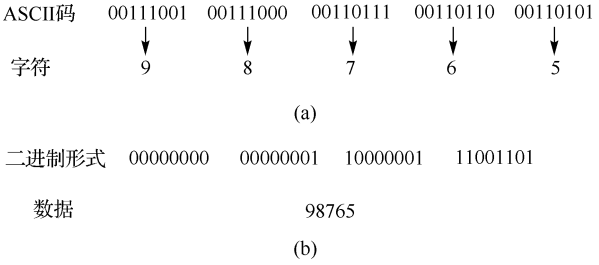


图 10-1　文本文件和二进制文件

C 语言处理文件时不区分类型，把文件看成一个字符或字节的序列，称为“数据流”，“数据流”是一个传输通道，数据可以从输入（标准输入、文件等）流入程序，称为输入流；也可以从程序流到输出（标准输出、文件等），称为输出流。输入流和输出流的开始和结束由程序控制而不受物理符号（如回车符）控制，因此数据文件也称“流式文件”。

对文件进行读/写操作是通过一个文件缓冲区完成的，如图 10-2 所示。当程序从磁盘中读取文件内容时，操作系统将文件内容的部分数据输入到输入缓冲区中，然后再将程序需要的数据传送给程序；当程序要将数据写入到文件中时，操作系统先将数据保存在输出缓冲区中，待一定条件满足时，才将缓冲区中的数据输出到磁盘的文件中。

C 语言使用一个 FILE 类型的指针变量指向一个文件，称为文件指针，通过文件指针可以对文件进行各种操作。定义文件指针的一般形式为：“FILE* 指针变量标识符;”，其中“FILE”为系统定义的一个结构体类型，包含在“stdio.h”中，存放了文件的相关信息。例如，“FILE* fp;”定义了一个指向 FILE 结构的指针变量，使用 fp 可以找到与关联的文件，从而对文件进行各种操作。通常称 fp 为指向一个文件的指针变量。

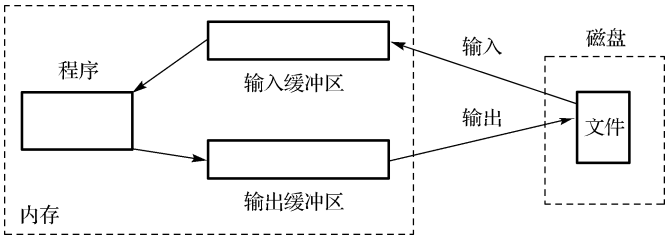


图 10-2　文件缓冲区

10.2　文件打开与关闭

对文件的操作都是由库函数来完成的。在进行文件读/写操作之前，要先打开文件，使用完后要关闭文件。打开文件是指建立文件的各种信息，将文件指针与该文件关联以进行其他操作；关闭文件是指断开指针与该文件的关联，此后禁止再对该文件进行操作。

10.2.1　文件打开

文件打开操作使用 fopen( ) 函数，其调用的一般形式为：“文件指针名=fopen（文件名,使用文件的方式）;”，其中，“文件指针名”是一个 FILE 类型的指针变量；“文件名”是将被打开的文件的名称，如果文件不在当前目录下，需要加上路径信息；“使用文件的方式”设置文件的类型和操作要求，由“r”（读）、“w”（写）、“a”（追加）、“b”（二进制文件，默认为文本文件）和“+”（读和写）5 个字符组成，共有 12 种，如表 10-1 所示。

表 10-1　使用文件的方式

使用文件的方式					符　号	意　义
文件操作			文件类型			
读	写	追加	文本文件	二进制文件		
√			√		r	打开一个文本文件，只允许读数据
	√		√		w	打开或新建一个文本文件，只允许写数据
		√	√		a	打开一个文本文件，在文件末尾写数据
√				√	rb	打开一个二进制文件，只允许读数据
	√			√	wb	打开或新建一个二进制文件，只允许写数据
		√		√	ab	打开一个二进制文件，在文件末尾写数据
√	√		√		r+	打开一个文本文件，允许读和写
√	√		√		w+	打开或新建一个文本文件，允许读和写
√		√	√		a+	打开一个文本文件，允许读或在文件末追加数据
√	√			√	rb+	打开一个二进制文件，允许读和写
√	√			√	wb+	打开或新建一个二进制文件，允许读和写
√		√		√	ab+	打开一个二进制文件，允许读或在文件末追加数据

例如，下列程序块中，打开当前目录下的文件 example.c，只允许进行读操作，并使 fp1 指向该文件；打开目录 test 下的 data 文件，允许读操作或在文件末追加数据，并使 fp2 指向该文件。

```
FILE *fp1,*fp2;
fp1=fopen("example.c", "r");
fp2=fopen("test/data", "ab+");
```

当用含“r”和“a”的方式打开文件时，该文件必须已经存在，如果该文件不存在，则打开失败出错；当用含“w”的方式打开文件时，若打开的文件不存在，则以指定的文件名新建一个文件，若指定的文件已经存在，则将该文件删去，重新建立一个新文件。打开文件时如果失败，fopen( ) 函数将返回一个空指针值 NULL，在程序中可以用来判断是否成功打开文件。例如，下列程序块中，如果在打开文件 data_file 时出错，则打印提示信息后终止程序运行。

```
if ((fp=fopen("data_file", "rb"))=NULL){
      printf("Error in open file!\n");
      return -1;
}
```

系统定义了三个标准的流文件：标准输入文件、标准输出文件、标准出错输出文件，并与终端设备进行关联，标准输入文件与标准输入设备（键盘）对应，标准输出文件和标准出错输出文件与标准输出设备（显示器）对应。程序开始运行时，系统自动打开这三个文件，并使用三个指针 stdin、stdout 和 stderr 分别指向标准输入文件、标准输出文件和标准出错输出文件，可以直接使用这三个指针进行标准输入和输出等操作。

10.2.2 文件关闭

文件使用后需要使用 fclose( ) 函数把文件关闭，以防止它再被误用，或者避免数据丢失，其调用的一般形式为：“fclose（文件指针）;”，成功关闭文件时，函数返回 0 值，如果出错，则返回结束标志 EOF（即–1）。例如，下列程序块中，fp 指向打开的文件 example.c，然后通过 fp 指针变量关闭文件，此后指针变量 fp 不再和该文件关联。

```
FILE *fp;
fp=fopen("example.c", "r");
fclose(fp);
```

10.3 文件基本操作

10.3.1 文件检测

在对文件进行操作时，常用的文件检测有文件结束检测、读写文件出错检测等。

（1）文件结束检测

使用 feof( ) 函数检测文件是否处于文件结束状态，一般形式为“feof（文件指针）;”，如文件已结束，则返回值 1，否则返回值 0。

（2）读写文件出错检测

使用 ferror( ) 函数检测文件在用各种输入/输出函数进行读写时是否有错误产生，一般形式为“ferror（文件指针）;”，如果没有错误产生，则返回值 0，否则返回非 0 值。

对同一个文件每次调用输入/输出函数都可能会产生错误，因此应当在调用一个输入/输出函数后使用 ferror( ) 函数进行检测。

此外，还有一个函数 clearerr( ) 用于将文件出错标志和文件结束标志置 0。一般形式为“clearerr（文件指针）;”。如果在调用输入/输出函数时产生了错误，调用 ferror( ) 函数返回一个非 0 值，此时应该再调用 clearerr( )函数，使 ferror( ) 函数值置为 0，以便下次检测。

10.3.2　顺序读/写文件

读文件和写文件是最常用的两种文件操作，头文件“stdio.h”中提供了 4 种文件读/写方式：字符方式（fgetc( ) 和 fputc( ) 函数）、字符串方式（fgets( ) 和 fputs( ) 函数）、格式化方式（fscanf( ) 和 fprintf( ) 函数）和数据块（二进制）方式（fread( ) 和 fwrite( ) 函数）。

1．字符方式

字符方式读/写文件是以字符为单位对文件进行读和写操作的，每次从文件中读取或向文件中写入一个字符。读文件函数为 fgetc( ) 函数，写文件函数为 fputc( ) 函数。

（1）fgetc( ) 函数

fgetc( ) 函数从指定文件中读取一个字符，格式为：“字符变量=fgetc（文件指针）;”，文件必须是以读或读写方式打开的。如果读操作成功，返回所读取的字符，赋值给字符变量；如果失败或到达文件末尾，则返回结束标志 EOF（即–1）。例如，语句“ch=fgetc(fp);”从 fp 指向的文件中读取一个字符并保存在 ch 中。

可以多次调用 fgetc( ) 函数读取多个字符。系统对文件维护了一个位置指针，指向文件中当前的读/写位置，文件打开时，该指针初始化指向文件开始位置（即第 1 字节），当调用一次 fgetc( ) 函数后，该指针向后方向移动 1 字节，下次调用 fgetc( ) 函数读取该位置处的数据。

【例 10-1】 程序 10-1：用 fgetc( ) 函数读入文件内容输出在屏幕上。

```
#01: //程序 10-1
#02: #include <stdio.h>
#03: #define LEN 32
#04: int main(){
#05:     char ch,filename[LEN];
#06:     FILE *fp;
#07:
#08:     printf("Input file to read:");
#09:     scanf("%s",filename);
#10:     if ((fp=fopen(filename,"r"))==NULL){
#11:         printf("Cannot open file %s.\n",filename);
#12:         return -1;
#13:     }
#14:
#15:     while ((ch=fgetc(fp))!=EOF)
#16:         putchar(ch);
#17:
#18:     printf("\n");
#19:     fclose(fp);
#20:     return 0;
#21: }
test.txt 文件内容：
abcdefg
hijklmn
opqrst
uvwxyz
```

程序解释：

#09：输入要读取的文件名。

#10～#13：只读方式打开文件 test.txt，如果出错，则退出程序。

#15：使用 fgetc( ) 函数读取字符，使用 while 循环读取文件中的所有字符，EOF 为文件末尾标志，值为–1。

程序运行结果如下：

```
Input file to read:test.txt
abcdefg
hijklmn
opqrst
uvwxyz
```

（2）fputc( ) 函数

fputc( ) 函数把一个字符写入到指定的文件中，格式为：“fputc（字符量，文件指针);”，其中“字符量”为将写入到文件的字符常量、变量或表达式。如果写入成功，函数返回值为写入的字符值；如果失败，则返回结束标志 EOF（即–1）。例如，语句“fputc('a', fp);”把字符'a'写入到 fp 指向的文件中。

被写入的文件可以用写、读写、追加等方式打开，用写或读写方式打开一个已存在的文件时，将清除文件原有内容，从文件开始写入字符。如果要保留文件原有内容，从文件末尾开始写入字符，则以追加方式打开文件。被写入的文件如果不存在，则创建一个文件。

与 fgetc( ) 函数相同，调用一次 fputc( ) 函数，文件内部的位置指针向后方向移动 1 字节，下次调用 fputc( ) 函数在新位置处写入字符数据。

【例 10-2】 程序 10-2：从屏幕上输入字符，用 fputc( ) 函数写入到文件中，直至遇到“.”字符结束。

```
#01: //程序 10-2
#02: #include <stdio.h>
#03: #define LEN 32
#04: int main(){
#05:     char ch,filename[LEN];
#06:     FILE *fp;
#07:
#08:     printf("Input file to write:");
#09:     scanf("%s",filename);
#10:     fflush(stdin);
#11:     if ((fp=fopen(filename,"w"))==NULL){
#12:         printf("Cannot open file %s.\n",filename);
#13:         return -1;
#14:     }
#15:
#16:     while ((ch=getchar())!='.')
#17:         fputc(ch,fp);
#18:
#19:     fclose(fp);
#20:     return 0;
#21: }
```

程序解释：

#09, #10：输入要写入的文件名，fflush(stdin)清除标准输入缓冲区。

#11～#14：只写方式打开文件 test.txt，如果出错，则退出程序。

#17：使用 fputc( ) 函数写入字符，使用 while 循环将所有字符写入到文件中。

程序运行结果如下：

```
Input file to write:test.txt
1234567890
qwertyuiop
asdfghjkl
zxcvbnm
.
text.txt 文件内容：
1234567890
qwertyuiop
asdfghjkl
zxcvbnm
```

2．字符串方式

字符串方式读/写文件是以字符串为单位对文件进行读和写操作，每次从文件中读取或向文件中写入一个字符串。读文件函数为 fgets( ) 函数，写文件函数为 fputs( ) 函数。

（1）fgets( ) 函数

fgets( ) 函数从指定的文件中读取一个字符串到字符数组中，格式为：“fgets（字符数组名,长度限制,文件指针）;”，其中“字符数组名”用于保存读取到的字符串，“长度限制”为一个整数 *n*，从文件中读取的字符串长度不超过 *n*–1。读取后自动在最后一个字符后面添加字符串结束标志 ‘\0’。如果读取成功，则返回字符数组的首地址，如果失败或到达文件末尾，则返回 NULL。例如，语句“fgets(str,n,fp);”从 fp 所指的文件中最多读取 *n*–1 个字符存入到字符数组 str 中。

对于 fgets( ) 函数，字符数组的长度要大于或等于 *n*（最多读取的 *n*–1 个字符和一个字符串结束标志 ‘\0’），如果在读取 *n*–1 个字符过程中，遇到了换行符（‘\n’）或文件结束符 EOF，则读取结束，其中换行符（‘\n’）也作为一个字符保存在字符数组中。

【例 10-3】 程序 3-3：用 fgets( ) 函数读入文件内容并输出在屏幕上。

```
#01: //程序 3-3
#02: #include <stdio.h>
#03: #define LEN 32
#04: #define SIZE 5
#05: int main(){
#06:     char ch,filename[LEN],buf[SIZE];
#07:     FILE *fp;
#08:
#09:     printf("Input file to read:");
#10:     scanf("%s",filename);
#11:     if ((fp=fopen(filename,"r"))==NULL){
#12:         printf("Cannot open file %s.\n",filename);
#13:         return -1;
```

```
#14:    }
#15:
#16:    while (fgets(buf,SIZE,fp)!=NULL)
#17:       printf("%s",buf);
#18:
#19:    printf("\n");
#20:    fclose(fp);
#21:    return 0;
#22: }
test.txt 文件内容：
abcdefg
hijklmn
opqrst
uvwxyz
```

程序解释：

#06：定义一个字符数组 buf[SIZE]，用来存放读取的文件内容，一次读取 SIZE–1 个字符。

#16～#17：使用 fgets( ) 函数读取字符串，使用 while 循环读取文件的所有内容，到达文件末尾时，返回 NULL。

程序运行结果如下：

```
Input file to read:test.txt
abcdefg
hijklmn
opqrst
uvwxyz
```

（2）fputs( ) 函数

fputs( ) 函数向指定的文件中写入一个字符串，格式为：“fputs（字符串,文件指针）;”，其中“字符串”为要写入文件的字符串，可以是字符串常量、字符数组名或指针变量等，字符串的结束标志‘\0’不写入到文件中。如果成功，则函数返回非负值，如果失败，返回结束标志 EOF（即–1）。例如，语句“fputs("abc", fp);”把字符串“abc”写入到 fp 所指向的文件中。

【例 10-4】 程序 10-4：从屏幕上输入字符，用 fputs( ) 函数写入到文件中，直至遇到“.”字符（单独一行）结束。

```
#01: //程序 10-4
#02: #include <stdio.h>
#03: #define LEN 32
#04: #define SIZE 5
#05: int main(){
#06:    char ch,filename[LEN],buf[SIZE];
#07:    FILE *fp;
#08:
#09:    printf("Input file to write:");
#10:    scanf("%s",filename);
#11:    fflush(stdin);
#12:    if ((fp=fopen(filename,"w"))==NULL){
```

```
#13:        printf("Cannot open file %s.\n",filename);
#14:        return -1;
#15:    }
#16:
#17:    while (strcmp(gets(buf),".")!=0){
#18:        fputs(buf,fp);
#19:        fputc('\n',fp);
#20:    }
#21:
#22:    fclose(fp);
#23:    return 0;
#24: }
```

程序解释：

#06：定义一个字符数组 buf[SIZE]，用来存放从标准输入读取的字符串，以存入文件中。

#12～#15：只写方式打开文件 test.txt，如果出错，则退出程序。

#17：使用 gets( ) 函数从标准输入读取字符串到 buf 中，buf 中不保存标准输入的回车符。使用 strcmp( ) 函数判断是否是“.”，以结束输入。

#18～#19：使用 fputs( ) 函数将 buf 内容写入到文件中，因为没有换行符，因此使用 fputc( ) 函数写入换行符。

程序运行结果如下：

```
Input file to write:test.txt
1234567890
qwertyuiop
asdfghjkl
zxcvbnm
.
text.txt 文件内容：
1234567890
qwertyuiop
asdfghjkl
zxcvbnm
```

3．格式化方式

格式化方式读/写文件是以指定的格式对文件进行读和写操作，读和写文件函数分别为 fscanf( ) 和 fprintf( ) 函数，与格式输入和格式输出函数 scanf( ) 和 printf( ) 功能类似，区别是 fscanf( ) 和 fprintf( ) 函数的读写对象不是标准输入（键盘）和标准输出（显示器），而是数据文件。

fscanf( ) 和 fprintf( ) 函数的格式为：“fscanf（文件指针,格式字符串,输入列表）;”和“fprintf（文件指针,格式字符串,输出列表）;”。例如，语句“fscanf(fp,"a=%d,str=%s",&a,str);”从 fp 所指向的文件中按照“a=整型变量,str=字符串”的格式读取整型变量和字符串的值；语句“fprintf(fp, "a=%d,str=%s", a,str);”将变量 a 和字符串 str 的值按照“a=整型变量,str=字符串”的格式保存在 fp 所指向的文件中。

【例 10-5】 程序 10-5：格式化方式读/写文件示例。

```
#01: //程序 10-5
#02: #include <stdio.h>
```

```
#03: #define LEN 32
#04: struct student{
#05:    int id;
#06:    char name[LEN];
#07:    char sex;
#08:    float score;
#09: };
#10: int main(){
#11:    struct student stu1,stu2,*pstu;
#12:    FILE *fp;
#13:
#14:    pstu=&stu1;
#15:    printf("Input stu id,name,sex,score:");
#16:    scanf("%d %s %c %f",&pstu->id,
              pstu->name,&pstu->sex,&pstu->score);
#17:
#18:    if ((fp=fopen("stu_data.txt","w+"))==NULL){
#19:       printf("Cannot open file!\n");
#20:       return -1;
#21:    }
#22:
#23:    fprintf(fp,"id=%d,name=%s ,sex=%c,score=%f\n",
               pstu->id,pstu->name,pstu->sex,pstu->score);
#24:    rewind(fp);
#25:    pstu=&stu2;
#26:    fscanf(fp,"id=%d,name=%s ,sex=%c,score=%f\n",
              &pstu->id,pstu->name,&pstu->sex,&pstu->score);
#27:
#28:    printf("Student info:\n");
#29:    printf("id\tname\tsex\tscore\t\n");
#30:    printf("%d\t%s\t%c\t%.1f\n",
              stu2.id,stu2.name,stu2.sex,stu2.score);
#31:
#32:    fclose(fp);
#33:    return 0;
#34: }
```

程序解释：

#14～#16：输入学生信息，保存在指针 pstu 指向的 stu1 变量中。

#18：用读写方式打开文件，如果文件存在，则清空文件。

#23：将 pstu 指向的 stu1 内容按照指定的格式写入到文件中。注意“name=%s”与“,sex=%c,score=%f\n”之间有一个空格，是为了读取内容时遇到空格结束 name 字符串的读取，否则，将后面的内容都作为 name 字符串的一部分。

#24：把 fp 所指文件的内部位置指针重新移到文件开始位置（以进行读操作）。

#25～#26：从文件中按照指定格式读取数据，保存在指针 pstu 指向的 stu2 变量中。注意“name=%s”与“,sex=%c”之间有一个空格，是为了匹配文件内容的格式。

#28～#30：输出结构体变量 stu2 的内容。

程序运行结果如下：

```
Input stu id,name,sex,score:1001 zhang3 m 98
Student info:
id     name    sex    score
1001    zhang3  m      98.0
stu_data.txt 文件内容:
id=1001,name=zhang3 ,sex=f,score=98.000000
```

4．数据块方式

数据块方式也称为二进制方式，该方式以二进制形式对文件进行读写，读/写单位是整块数据，如一个或多个数组、结构体等。读文件时，将文件中若干字节的内容一批读入到内存中；写文件时，直接将内存中一组数据原封不动地写入文件中。

读数据块函数为 fread( )，写数据块函数为 fwrite( )，格式为：“fread（字符数组名,数据块字节数,数据块个数,文件指针）;”和“fwrite（字符数组名,数据块字节数,数据块个数,文件指针）;”，其中“字符数组名”用来保存读入的数据或要写入文件中的数据，也可以为指针变量；“数据块字节数”指定一个数据块的大小；“数据块个数”指定读/写多少个数据块。如果读写成功，则返回读/写的数据块个数，如果失败或到达文件末尾，则返回实际读/写的数据块个数或 0。例如，语句“fread(array,4,10,fp);”从 fp 指向的文件中读取 10 个 4 字节大小（int 类型）的数据，存入到 array 数组中。

【例 10-6】 程序 10-6：数据块方式读取文件示例。

```
#01: //程序 10-6
#02: #include <stdio.h>
#03: #define LEN 32
#04: #define N 3
#05: struct student{
#06:     int id;
#07:     char name[LEN];
#08:     char sex;
#09:     float score;
#10: };
#11: int writefile(struct student*,int,FILE *);
#12: int readfile(struct student*,int,FILE *);
#13:
#14: int main(){
#15:     struct student stu[N];
#16:     FILE *fp;
#17:
#18:     if ((fp=fopen("stu_data","wb+"))==NULL){
#19:        printf("Cannot open file!\n");
#20:        return -1;
#21:     }
#22:
#23:     writefile(stu,N,fp);
```

```
#24:     rewind(fp);
#25:     readfile(stu,N,fp);
#26:
#27:     fclose(fp);
#28:     return 0;
#29: }
#30:
#31: int writefile(struct student* pstu,int num,FILE *fp){
#32:     struct student *ps;
#33:     for(ps=pstu;ps<pstu+num;ps++){
#34:        printf("Input %d student's id,name,sex,score:",ps-pstu);
#35:        scanf("%d %s %c %f",&ps->id,ps->name,&ps->sex,&ps->score);
#36:        fflush(stdin);
#37:     }
#38:
#39:     if (fwrite(pstu,sizeof(struct student),num,fp)!=num){
#40:        printf("Error in writing file.\n");
#41:        return -1;
#42:     }
#43:     return 0;
#44: }
#45:
#46: int readfile(struct student* pstu,int num,FILE *fp){
#47:     struct student *ps;
#48:
#49:     if (fread(pstu,sizeof(struct student),num,fp)!=num){
#50:        printf("Error in reading file.\n");
#51:        return -1;
#52:     }
#53:
#54:     printf("Student info:\n");
#55:     printf("id\tname\tsex\tscore\t\n");
#56:     for(ps=pstu;ps<pstu+num;ps++){
#57:        printf("%d\t%s\t%c\t%.1f\n",
                    ps->id,ps->name,ps->sex,ps->score);
#58:     }
#59:    return 0;
#60: }
```

程序解释：

#18：使用读写方式和二进制方式打开文件。

#23～#24：先写入数据到文件中，然后使文件中的位置指针重新指向文件开始位置，再从文件中读取数据。

#31～#44：从标准输入读取数据存入 pstu 指向的结构体数组中，然后使用 fwrite( ) 函数一次将结构体数组内容写入到文件 fp 中。

#46～#60：先使用 fread( ) 函数一次将文件 fp 内容读到 pstu 指向的结构体数组中保存，然后将结构体数组中的数据显示在标准输出上。

程序运行结果如下：

```
Input 0 student's id,name,sex,score:1001 zhang3 m 95
Input 1 student's id,name,sex,score:1002 li4 f 98
Input 2 student's id,name,sex,score:1003 wang5 m 88
Student info:
id     name    sex    score
1001    zhang3  m      95.0
1002    li4     f      98.0
1003    wang5   m      88.0
```

10.3.3　随机读/写文件

顺序读/写文件只能从文件开始位置读/写数据。当文件较大时，通常只需要读/写文件中某一部分数据，为此，可以先将文件内部的位置指针移动到需要读/写的位置，然后再进行文件读/写，称为随机读/写。

随机读/写并不按照数据在文件中的物理位置顺序进行读/写，而是可以对文件内部任意位置上的数据进行读/写。实现随机读/写的关键是移动位置指针到适当位置，称为文件定位。

系统为每个打开的文件维护了一个位置指针，用来指示当前要读/写的位置。文件打开时，该指针初始化指向文件头部（即开始位置，文件第 1 字节）。当进行读或写 *n* 字节操作时，系统从位置指针处读取或写入 *n* 字节，并将位置指针向后移动 *n* 字节，下次从新的位置处读/写数据。直至遇到文件结尾，读操作结束，写操作可以继续写入（将使文件大小变大）。如图 10-3 所示。

可以根据需要移动位置指针，从而改变当前的读/写位置，可以向前、向后移动 *n* 字节到新的位置，实现对文件的随机读/写。总之，通过位置指针可以在文件内部的任意位置处读取或写入数据。

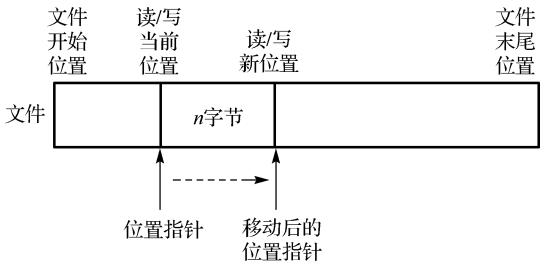


图 10-3　文件位置指针

有三个移动文件内部位置指针的函数：rewind( ) 函数、fseek( ) 函数和 ftell( ) 函数。

rewind( ) 函数把文件内部的位置指针重新移到文件开始位置，格式为“rewind（文件指针）;”，当需要重新从文件开始位置读/写数据时，可以调用该函数。

fseek( ) 函数按要求移动位置指针，一般用于二进制文件。格式为“fseek（文件指针,位移量,起始参考点）;”，其中“位移量”设置移动的字节数，以“起始参考点”为基点。该参数类型为 long 型，用常量表示位移量时，后面应加后缀“L”。“位移量”如果为正值，则向后移动位置指针（文件末尾方向）；如果为负值，则向前移动位置指针（文件开始方向）。“起始参考点”指定从何处开始计算位移量，一共有三种：文件开始、当前位置和文件末尾。如表 10-2 所示。

例如，语句“fseek(fp,100L, SEEK_SET);”将位置指针移到离文件开始位置 100 字节处；语句“fseek(fp,–100L, SEEK_END);”将位置指针移到离文件结束位置 100 字节处。

表 10-2 fseek( ) 函数起始参考点

起始参考点符号	意 义	对应的数值
SEEK_SET	文件开始位置	0
SEEK_CUR	文件当前位置	1
SEEK_END	文件末尾位置	2

ftell( ) 函数得到文件内部位置指针的当前值，格式为“ftell（文件指针）;”，调用该函数返回一个 long 型的相对文件开始位置的位移量，如果失败则返回–1。

【例 10-7】 程序 10-7：随机读/写文件示例，从例 10-6 建立的文件中读取第二个学生的信息。

```
#01: // 程序 10-7
#02: #include <stdio.h>
#03: #define LEN 32
#04: #define NO 1
#05: struct student{
#06:     int id;
#07:     char name[LEN];
#08:     char sex;
#09:     float score;
#10: };
#11: int main(){
#12:     struct student stu;
#13:     FILE *fp;
#14:
#15:     if ((fp=fopen("stu_data.txt","rb"))==NULL){
#16:        printf("Cannot open file!\n");
#17:        return -1;
#18:     }
#19:
#20:     fseek(fp,sizeof(struct student)*NO,SEEK_SET);
#21:     if (fread(&stu,sizeof(struct student),1,fp)!=1){
#22:        printf("Error in reading file.\n");
#23:        return -1;
#24:     }
#25:
#26:     printf("No.%d Student info:\n",NO);
#27:     printf("id\tname\tsex\tscore\t\n");
#28:     printf("%d\t%s\t%c\t%.1f\n",
                stu.id,stu.name,stu.sex,stu.score);
#29:
#30:     fclose(fp);
#31:     return 0;
#32: }
```

程序解释：

#04：定义要输出信息的学生编号，1 为第二个学生。

#15：以只读方式打开二进制文件。

#20：将位置指针移到到第二个学生的位置（相对于文件开始位置，向后移动一个 student 结构体的大小）。

#21～#24：读取一个结构体大小的数据块，存放在&stu 地址开始的位置。

#26～#28：输出结构体变量 stu 中的数据。

程序运行结果如下：

```
No.1 Student info:
id     name    sex     score
1002    li4     f      98.0
```

10.4　程 序 示 例

【例 10-8】 程序 10-8：将一个文件（源文件）的最后 30 字节复制到另外一个文件（目的文件）中，如果目的文件名为 out，则将内容输出到标准输出文件（stdout）中。

```
#01: //程序 10-8
#02: #include <stdio.h>
#03: #define LEN 32
#04: #define SIZE 5
#05: #define OFFSET 30L
#06: int main() {
#07:     FILE *fp_src, *fp_dest;
#08:     char src[LEN],dest[LEN],buf[SIZE];
#09:     int len;
#10:
#11:     printf("Input src file to read:");
#12:     scanf("%s",src);
#13:     if ((fp_src=fopen(src,"r"))==NULL){
#14:        printf("Cannot open file %s.\n",src);
#15:        return -1;
#16:     }
#17:
#18:     printf("Input dest file to write:");
#19:     scanf("%s",dest);
#20:     if (strcmp(dest,"out")==0)
#21:        fp_dest=stdout;
#22:     else if ((fp_dest=fopen(dest,"w"))==NULL){
#23:        printf("Cannot open file %s.\n",dest);
#24:        return -1;
#25:     }
#26:
#27:     fseek(fp_src, -OFFSET, SEEK_END);
#28:
#29:     while((len=fread(buf,sizeof(char),sizeof(buf),fp_src))>0){
#30:         if(fwrite(buf,sizeof(char),len,fp_dest)!=len){
#31:             printf("Write dest file error!\n");
#32:            return -1;
```

```
#33:         }
#34:     }
#35:
#36:     fclose(fp_src);
#37:     if (strcmp(dest,"out")!=0)
#38:        fclose(fp_dest);
#39:     return 0;
#40: }
```

程序解释：

#05：宏定义位移量 OFFSET，指定为 long 型。

#11～#16：输入将读取数据的源文件名，并用只读方式打开。

#18～#25：输入将写入数据的目的文件名，并用只写方式打开。如果输入“out”，则将目的文件指针设置为标准输出文件 stdout。

#27：将文件内位置指针移动到距离文件末尾 30 字节处。

#29～#34：将位置指针到文件末尾的数据写入到目的文件中，fread( ) 的数据大小为 1 字节，一次读取 sizeof(buf) 个数据。如果到达文件末尾，则返回 0，如果数据不足 sizeof(buf)个，则返回实际读取到的数据个数（存入变量 len）。fwrite( )写入数据时，写入的数据个数为实际读取到的字节数（len）。

#37,#38：如果目的文件指针不是标准输出文件，则将其关闭。

如果文件为文本文件，则显示的字符数可能小于 30，这是由于是按照二进制 fread( )/fwrite( )的方式读取和写入，文本文件中换行符被替换成两个字节“0x0D 0x0A”。

程序运行结果如下：

```
Input src file to read:test.txt
Input dest file to write:out
ertyuiop
asdfghjkl
zxcvbnm
```

上机实验：文件程序设计应用

本次实验掌握 C 语言程序的文件用法，理解文件指针的用法，熟练掌握文件的打开和关闭方法、各种方式对文件的读/写。

（1）回显屏幕，即将输入的字符串重新输出在屏幕上。使用 fgetc( )/fputc( )函数和 fgets( )/fputs( )函数两种方式实现。

程序示例：

① fgetc( )/fputc( )函数方式：

```
#include <stdio.h>
int main() {
    int ch;

    while ((ch=fgetc(stdin))!=EOF)
        if (fputc(ch,stdout)==EOF){
            printf("Echo screen error!\n");
```

```
                return -1;
            }

        if (ferror(stdin)){
            printf("Read screen error!\n");
            return -1;
        }
        return 0;
    }
```

② fgets( )/fputs( ) 函数方式：

```
    #include <stdio.h>
    #define SIZE 128
    int main(){
        char buf[SIZE];

        while ((fgets(buf,SIZE,stdin))!=NULL)
            if (fputs(buf,stdout)==EOF){
                printf("Echo screen error!\n");
                return -1;
            }

        if (ferror(stdin)){
            printf("Read screen error!\n");
            return -1;
        }
        return 0;
    }
```

（2）将一个文本文件中的小写字母转换为大写字母，另存为一个文件，文件名为 changed.txt。程序示例：

```
    #include <stdio.h>
    #define LEN 32
    int main(){
       char ch,filename[LEN];
       FILE *fps,*fpd;

       printf("Input file to read:");
       scanf("%s",filename);
       fps=fopen(filename,"r");
       fpd=fopen("changed.txt","w");

       if(fps==NULL||fpd==NULL){
            printf("Open file error!\n");
            return -1;
        }
```

```
    while ((ch=fgetc(fps))!=EOF){
        if ((ch>='a')&&(ch<='z'))
            ch-=32;
        else if ((ch>='A')&&(ch<='Z'))
            ch+=32;
        fputc(ch,fpd);
    }

    fclose(fps);
    fclose(fpd);
    return 0;
}
```

习　题

1．有两个文本文件，将第二个文件的内容追加到第一个文件的末尾。

2．将一个文本文件的每一行前加行号。

3．从键盘输入字符串，将其中的大写字母转换为小写字母，并保存在文件“lower.txt”中，输入以“.”结束。然后将文件“lower.txt”中的内容读出显示在屏幕上。

4．定义一个结构体类型 Data（年、月、日），将 *N* 个 Data 变量存入文件中。然后读取第奇数个日期并输出。

5．将两个文本文件的内容隔行存入到新文件中（奇数行为第一个文件的第一行，偶数行为第二个文件的内容）。

6．将格式化的数据（*N* 个学生信息：学号、姓名、性别、三门课程成绩）写入到文件中，再从该文件中以格式化方法读出数据并输出。

7．输入 *N* 个整数，存入二进制文件中，并读取这些整数，统计其中正整数、零和负整数的个数。

8．输入 *N* 个字符串，将其存入到文件中，然后对字符串进行排序，将排序后的字符串存入另一个文件中。

第 11 章　高质量编程规范

为了规范程序设计，提高 C 语言程序的质量，使得程序更高效、易扩展、减少出错，本章摘录了由电子工业出版社出版的林锐博士编著的《高质量程序设计指南——C++/C 语言（第 3 版）》一书相关章节并进行了修改，使之更适合初学者。

11.1　宏观上高质量

11.1.1　编码的风格

1．头文件结构

每个 C 程序通常分为两个文件：一个用于程序声明，称为头文件，后缀名为“.h”；另一个用于程序实现，称为“定义文件”，后缀名为“.c”。如果一个软件系统的头文件和定义文件比较多，为了便于维护，可以将头文件和定义文件分别保存在不同目录中，例如，将头文件保存在“include”目录，将定义文件保存在“source”目录。

头文件包括三部分内容：①注释部分，描述文件作用、版本等；②编译预处理部分；③函数和变量声明等。例如，下面程序块中给出了“example.h”头文件的内容。

```
//文件作用、版本等描述
#ifndef EXAMPLE H    // 防止 example.h 被重复引用
#define EXAMPLE H

#include <stdio.h>   // 引用标准库的头文件
#include "myfile.h"  // 引用非标准库的头文件
void function1(…);  // 全局函数声明

#endif
```

对于头文件，应遵守如下规则：

（1）为了防止头文件被重复引用，应使用“ifndef/define/endif”结构形成预处理块；

（2）使用“#include <filename.h>”格式引用标准库的头文件；使用“#include "filename.h"”格式引用用户自己的头文件；

（3）头文件中只存放“声明”而不存放“定义”；

（4）不提倡使用全局变量，尽量不要在头文件中使用“extern”声明。

2．定义文件结构

定义文件包括三部分内容：①注释部分，描述文件作用、版本等；②对其他头文件的引用；③程序的实现部分（包括数据和代码）。例如，下面程序块中给出了“example.c”定义文件的内容。

```
//文件作用、版本等描述
#include “example.h”//引用头文件
void function(){  // 函数实现
}
```

3．标识符命名

在程序设计过程中，保持变量函数等命名的统一，能有效增强程序的可读性。常用的代码书写规范有三种：匈牙利命名法、骆驼命名法和帕斯卡命名法。这些方法各有优、缺点，在程序设计时应遵守同一个规范，除此之外应遵守以下规则。

（1）标识符应当直观且可以拼读，可以“望文知意”，不必进行“解读”。最好采用英文单词或组合，便于记忆和阅读，切忌使用汉语拼音来命名。例如，不要把“currentValue”写成“nowValue”。

（2）标识符的长度应当符合“最短长度最大信息”原则，避免太长和太短的标识符。例如，不要命名为“maxScoreValue”，而用“maxVal”。单字符变量 i、j、k 等通常用作循环变量或复合语句内的局部变量。

（3）命名规则尽量与所采用的操作系统或开发工具的风格保持一致，不要混在一起使用。例如，Windows 应用程序的标识符通常采用“大小写”混排的方式，如 AddChild；而 Unix 应用程序的标识符通常采用“小写加下画线”的方式，如 add_child。

（4）程序中不要出现仅靠大小写区分的相似的标识符，或者不易区分的标识符。如“x”和“X”，字母“O”和数字“0”、字母“l”和数字“1”等。

（5）程序中不要出现与标识符完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但会使人误解。

（6）变量名称应使用“名词”或“形容词+名词”形式。如“oldValue”。

（7）函数名称应使用“动词”或“动词＋名词”形式。如“findMax( )”。

（8）用正确的反义词组命名具有互斥意义的变量或相反动作的函数，如“minValue”、“maxValue”等。

（9）尽量避免名称中出现数字编号，如 Value1、Value2 等，除非逻辑上需要编号。

11.1.2 程序的版式

好的程序版式可以使程序易读，方便调试。应从以下几方面注意程序的书写。

1．注释

C 语言的注释符号为“//”和“/*…*/”，程序块的注释用“/*…*/”，行注释用“//…”。注释可用于文件的功能描述、版本和版权声明、函数接口说明、重要的代码行或段落说明等。注释遵守以下规则。

（1）注释是对代码的“提示”，而不是文档。程序中的注释不可太多，否则会让人眼花缭乱，影响阅读程序。

（2）如果代码本来就是清楚的，则不必加注释。尽量避免在注释中使用缩写，特别是不常用的缩写。

（3）边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

（4）注释应当准确、易懂，防止注释有二义性。错误的注释不但无益，反而有害。

（5）注释的位置应与被描述的代码相邻，可以放在代码的上方或右方，不可放在下方。当代码比较长，特别是有多重嵌套时，应当在一些段落的结束处加注释，便于阅读。

2．空行

空行起着分隔程序段落的作用，可以使程序的布局更加清晰。使用空行时应注意以下规则。

（1）每个函数定义结束之后都要加空行。

（2）在一个函数体内，逻辑上密切相关的语句之间不加空行，功能上较为独立的语句片段之间应加空行分隔。

3．代码

一行代码只做一件事情，例如，一行代码只定义一种类型的变量，如“int i, sum;”，不要定义多种类型的变量，如“int i，*ptr;”。这样的代码容易阅读，并且方便书写注释。对于代码行，应注意以下事项。

（1）对于 if、for、while、do 等语句独立占一行，执行语句另起一行，不论执行语句有多少，都要加“{}”以防止书写失误。

（2）定义变量的同时初始化该变量，否则容易遗漏初始化，导致程序出错。

（3）关键字后面要有空格，如 if、for、while 等关键字之后留一个空格再跟左括号“(”，可以突出关键字。函数名之后不要留空格，紧跟左括号“(”，以区别关键字。

（4）左括号“(”与后面代码之间，右括号“)”、逗号“,”和分号“;”与前面代码之间不留空格。逗号“,”与后面代码之间要留空格，例如，“fun(x, y, z)”。如果分号“;”不是一行的结束符号，后面要留空格，如“for (initialization; condition; update)”。

（5）赋值运算符、比较运算符、算术运算符、逻辑运算符，如“=”、“+=”、“>=”、“<=”、“+”、“*”、“%”、“&&”、“||”、“<<”和“^”等二元运算符的前后应当加空格。一元运算符，如“!”、“～”、“++”、“—”、“&”（地址运算符）等前后不加空格。像“[]”、“. ”、“→”等运算符前后不加空格。

（6）对于表达式比较长的 for 语句和 if 语句，为了紧凑起见，可以适当地去掉一些空格，如“for (i=0; i<10; i++)”、“if ((a<=b) && (c<=d))”等。

（7）程序的分界符“}”应单独占一行，与引用它的语句左对齐；“{”可以放在上一条语句后面。“{}”之内的代码块应缩进一层。

（8）代码行最大长度应控制在 70～80 个字符以内。较长的表达式要在低优先级运算符处拆分成新行，运算符放在新行之首（以便突出运算符）。拆分出的新行要进行适当缩进，使排版整齐，语句可读。例如，下列程序块中的语句被拆成多行。

```
if ((very_longer_variable1 >= very_longer_variable12)
    && (very_longer_variable3 <= very_longer_variable14)
    && (very_longer_variable5 <= very_longer_variable16)){
      dosomething();
}
for (very longer initialization;
    very longer condition;
    very longer update){
    dosomething();
}
```

（9）将修饰符“*”和“＆”紧靠变量名，如“int *x, y;”，这样变量 y 就不会被误解成指针变量。

11.2　微观上高质量

11.2.1　程序的健壮性

1．使用断言

程序一般分为 Debug 版本和 Release 版本，Debug 版本用于内部调试，Release 版本发行给用户使用。断言 assert 是仅在 Debug 版本中起作用的宏，它用于检查“不应该”发生的情况。下列程序段是一个字符串复制函数，如果 assert 的参数为假，那么程序就会中止。

```
char *strcpy(char *strDest, const char *strSrc);{
   assert((strDest!=NULL) && (strSrc !=NULL));
   char *address = strDest;
   while( (*strDest++ = * strSrc++) != '\0' )
     ;
   return address ;
}
```

使用断言，应遵守如下规则：

（1）使用断言捕捉不应该发生的非法情况。不要混淆非法情况与错误情况之间的区别，后者是必然存在的，并且是一定要做出处理的；

（2）在函数的入口处，使用断言检查参数的有效性（合法性）；

（3）在编写函数时，要进行反复的考查，并且自问："我打算做哪些假定？"，一旦确定了假定，就要使用断言对假定进行检查。

2．表达式

如果表达式中的运算符比较多，为了防止产生歧义并提高可读性，应当用括号确定表达式的操作顺序，避免使用默认优先级。如"word = (high<<8)|low"、"if ((a|b) && (a&c))"。

复合表达式书写简洁，可以提高编译效率，如"a = b = c = 0"。但不要编写太复杂的复合表达式。例如，"i = a >= b && c < d && c + f <= g + h ;"应该避免。

也不要有多用途的复合表达式。例如，"d = (a = b + c) + r ;"应该拆分为两个独立的语句："a = b + c;"和"d = a + r;"。

不要把程序中的复合表达式与"数学上的表达式"混淆。例如，"if (a < b < c)"并不表示"if ((a<b) && (b<c))"，而是成了令人费解的"if ((a<b)<c)"。

3．if 和 switch 语句

if 语句与零值进行比较的规则如下。

（1）整型变量与零值比较时，将整型变量用"=="或"!="直接与 0 比较，如"if (value == 0)"或"if (value != 0)"，不要写成"if (value)"或"if (!value)"，否则会误解 value 的类型。

（2）浮点类型变量与零值比较时，不能将浮点变量用"=="或"！="与任何数字比较，因为浮点类型有精度限制，应转化成">="或"<="形式。例如，将错误的"if (x == 0.0)"转换为"if ((x>=–EPSINON) && (x<=EPSINON))"，其中 EPSINON 是允许的误差（精度）。

（3）指针变量与零值比较时，将指针变量用"=="或"!="与 NULL 比较，尽管 NULL 的值与 0 相同，但是两者意义不同。例如，"if (p == NULL)"或"if (p != NULL)"，不要写成"if (p == 0)"或"if (p != 0)"，否则会误解 p 为整型变量。

为了防止将"if (a ==常量)"误写成"if (常量= a)"，可以写成"if (常量== a)"的形式，将变量 a 和常量颠倒，这样如果误写，编译器会报错。

对于"if/else/return"组合，不要写成如下的程序：

```
if (condition)
     return x;
return y;
```

而应该写成如下的程序，或者"return (condition ? x : y);"。

```
if (condition){
     return x;
}else{
     return y;
}
```

对于 switch 语句，每个 case 语句的结尾都不要忘了加 break 语句，否则将导致多个分支重叠，除非有意使多个分支重叠。不要忘记最后的 default 分支，即使程序不需要 default 处理，也应该保留语句“default : break;”，这样可以防止被误解为忘记处理 default 情况。

4．for 语句

不能在 for 循环体内修改循环变量，防止 for 循环失去控制。对于 for 语句的循环控制变量的取值，采用“半开半闭区间”形式，例如，“for (int x=0; x<N; x++)”，不要写成“for (int x=0; x<=N–1; x++)”。

只在如下情况使用 goto 语句：从多重循环体中跳转到最外面，这样不用写很多 break 语句。

11.2.2　程序的优化

1．常量

尽量使用含义直观的常量来表示那些在程序中多次出现的数字或字符串。C 语言可以用 const 或 #define 来定义常量，前者比后者更有优点，const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查，而对后者只进行字符串替换，没有类型安全检查，并且在字符串替换时可能会产生错误。

如果某一常量与其他常量有关联，应在定义中包含这种关系，如“const float RADIUS=100;”和“const float DIAMETER=RADIUS * 2;”。

2．循环体效率

提高循环体效率的基本办法是降低循环体的复杂性，应遵守如下规则。

（1）在多重循环中，尽可能将最长的循环放在最内层，最短的循环放在最外层，以减少 CPU 跨切循环层的次数。例如，下列程序段中程序 b 的效率比程序 a 的高。

程序段 a：

```
for (row=0; row<100; row++){
    for ( col=0; col<5; col++ ){
        sum = sum + a[row][col];
    }
}
```

程序段 b：

```
for (col=0; col<5; col++ ){
    for (row=0; row<100; row++){
        sum = sum + a[row][col];
    }
}
```

（2）如果循环体内存在逻辑判断，并且循环次数很大，宜将逻辑判断移到循环体的外面，例如，下列程序段中，程序 a 比程序 b 多执行了 $N-1$ 次逻辑判断。而且由于前者要进行逻辑判断，打断了循

环“流水线”作业，使得编译器不能对循环进行优化处理，降低了效率。如果 *N* 非常大，最好采用程序 b 的写法，可以提高效率。如果 *N* 非常小，两者效率差别并不明显，采用程序 a 的写法比较好，因为程序更加简洁。

程序段 a：

```
for (i=0; i<N; i++){
    if (condition)
        DoSomething();
    else
        DoOtherthing();
}
```

程序段 b:

```
if (condition){
    for (i=0; i<N; i++)
        DoSomething();
}else{
    for (i=0; i<N; i++)
    DoOtherthing();
}
```

11.2.3 函数设计

函数的功能要单一，不要设计多用途的函数。函数体尽量控制在几十行代码之内。尽量少用 static 局部变量，避免函数带有“记忆”功能。对于函数，相同的输入应当产生相同的输出，带有“记忆”功能的函数，不易理解又不利于测试和维护。

1．参数

对于函数参数，应遵守如下规则。

（1）不仅要检查形式参数的有效性，还要检查其他进入函数体内的变量的有效性，如全局变量、文件指针等。

（2）参数的书写要完整，不能只写参数的类型而省略参数名字，如果函数没有参数，应使用“void”，如“float GetValue(void)”。

（3）参数命名要恰当，顺序要合理。避免函数有过多的参数，个数尽量控制在 5 个以内。如果参数过多，容易将参数类型或顺序混淆。

（4）如果参数是指针，且仅作为输入用，则应在类型前加 const，以防该指针在函数体内被意外修改。例如：“void StringCopy(char *strDestination，const char *strSource);”。

2．返回值

对于函数返回值，应遵守如下规则。

（1）不要省略返回值的类型。C 语言中凡是不加类型说明的函数，一律自动按整型处理，否则容易被误解为 void 类型。

（2）函数名字与返回值类型在语义上不可冲突。如“getMaxInt( )”函数的返回值类型应该为 int 型。

（3）有时函数不需要返回值，为了增加灵活性如（支持链式表达式），可以增加一个返回值。例

如，字符串复制函数“char *strcpy(char *strDest，const char *strSrc);”，函数的返回值是 strDest，这样可以支持链式表达式：“int length=strlen(strcpy(str, “Hello World”));”。

3．函数体

对于函数体，应遵守如下规则。

（1）在函数体的“入口处”对参数有效性进行检查。使用断言（assert）来防止非法参数错误。在函数体的“出口处”，对 return 语句的正确性和效率进行检查。

（2）不能使用 return 语句返回指向“栈内存”的“指针”，因为该内存在函数体结束时被自动销毁。例如：

```
char * func(void){
char str[] =”hello world”;   //str 的内存位于栈上
return str;      //将导致错误
}
```

11.2.4　指针

1．指针与数组

指针和数组在一些地方可以相互替换，但是两者是有区别的。数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。数组名对应着一块内存，其地址与长度在生命期内保持不变，只有数组的内容可以改变；指针可以随时指向任意类型的内存块，是“可变”的，可以操作动态内存。指针和数组在对字符串进行处理时存在以下不同。

（1）修改内容。下列程序块中，字符数组 a 的内容可以改变，如 a[0]='X'。指针 p 指向静态存储区的常量字符串“world”，内容是不可修改的，试图修改常量字符串的内容，将导致运行错误。

```
char a[] = “hello”;
a[0] = ‘X’;
printf(“%s\n”,a);
char *p = “world”;     // 注意 p 指向常量字符串
p[0] =’X’;            // 编译器不能发现该错误
printf(“%s\n”,p);
```

（2）内容复制与比较。不能对数组名进行直接复制与比较，应使用 strcpy( )和 strcmp( )函数。下列程序块中，语句“p = a”不能把 a 的内容复制给指针 p，只能把 a 的地址赋给 p。如果要复制内容，可以先用函数 malloc( )为 p 申请一块容量为 strlen(a)+1 个字符的内存，再用 strcpy( )进行复制。同理，语句“if(p==a)”比较的不是内容而是地址，应用函数 strcmp( )进行比较。

```
char a[] = "hello";
char b[10];
strcpy(b, a);
if(strcmp(b, a) == 0)

char *p = (char *)malloc(sizeof(char)*(strlen(a)+1));
strcpy(p,a); // 不要用 p = a;
if(strcmp(p, a) == 0) // 不要用 if (p == a)
```

（3）计算内存容量。用运算符 sizeof 可以计算出数组的容量（字节数），对于指针，sizeof ( )得到的

是一个指针变量的字节数而不是指针所指的内存容量。下列程序块中，sizeof(p)相当于 sizeof(char*)。当数组作为函数的参数时，该数组自动退化为同类型的指针，sizeof(b)始终等于 sizeof(char *)。

```
char a[] = "hello world";
char *p  = a;
printf("%d\n",sizeof(a));// 12 字节，注意字符串最后有一个'\0'
printf("%d\n",sizeof(p));// 4 字节

void Func(char b[100]){
    printf("%d\n",sizeof(b));// 4 字节而不是 100 字节
}
```

2．野指针

“野指针”不是 NULL 指针，是指向“垃圾”内存的指针，if 语句对“野指针”不起作用。“野指针”的成因主要有以下两种。

（1）指针变量没有被初始化。任何指针变量刚被创建时不会自动被初始化为 NULL，它的默认值是随机的。所以，指针变量在创建的同时应当被初始化，要么将指针设置为 NULL，要么让它指向合法的内存。例如，“char *p = NULL;”，“char *str = (char *) malloc(100);”。

（2）指针 p 被 free 后没有置为 NULL。

另外，避免数组或指针的下标越界，特别要当心发生“多 1”或 “少 1”的操作。

附录 A　C 语言课程设计

A.1　目　　的

C 语言在计算机软件开发中有着重要的地位，为了巩固对 C 语言理论知识的学习和理解，掌握 C 语言相关内容，提高使用 C 语言编程解决实际问题的能力，完成一个较为系统的课程设计非常必要。通过 C 语言课程设计，应达到如下目的：

（1）提高用程序设计解决实际问题的能力；

（2）掌握项目设计、实现、调试的基本流程；

（3）提高代码的编写能力和调试能力；

（4）评测程序是否运行正常、是否满足设计指标，并评价其效率。

A.2　课程设计流程

进行 C 语言课程设计的流程如下。

（1）分析问题。分析课程设计的要求，把较大问题分解成多个较小的问题，使用自顶向下的设计方法进行模块化。

（2）设计算法完成特定功能。使用伪代码或流程图实现各个模块的功能。

（3）实现算法。使用 C 语言将算法实现。

（4）测试每个模块是否满足要求，并修改错误的地方。

（5）将各个模块合并起来，完成整个软件，并对程序进行调试，测试程序是否有 bug，对程序进行完善。

（6）完成用户文档和用户手册，对软件的使用进行说明。

（7）完成技术文档，对程序中主要算法或模块进行说明，并提供一个完整的程序流程图。

A.3　要　　求

为了衡量学生掌握和运用 C 语言知识的水平和能力，应要求学生提供如下资料，以对学生进行评测。

（1）程序源代码和可执行程序。

（2）课程设计报告，内容应包括：①总体设计和算法分析与流程图；②详细设计、模块功能说明、函数功能描述等；③测试数据和测试过程记录，测试结果分析与讨论；④遇到的问题及解决方法。

（3）用户文档和技术文档。

A.4　评　　测

评测是检测学生是否掌握该门课程的重要手段，检查学生是否能够使用 C 语言来解决问题，达到课程的教学目的。教师根据学生提交的资料，检查完成情况：

（1）程序

源代码程序结构清晰，模块划分合理，程序流程清晰，注释充足。程序能够正常运行，测试结果正确。

（2）设计报告

课程设计报告内容完整，问题陈述正确，解决方案合理，采用自顶向下模块化设计方法，算法正确，测试数据充足合理。

（3）文档

用户文档信息齐全，技术文档详尽、准确。

A.5 项 目 参 考

A.5.1 学生管理系统

1. 要求

实现一个简单的学生管理系统，管理学生三门课成绩，实现以下功能：

（1）实现操作界面，通过一个简单的菜单选择相应的操作；

（2）录入或添加学生基本信息，学号不能重复；

（3）删除学生信息；

（4）修改学生信息，指定学号，修改该学生的成绩；

（5）查找指定的学生，可根据学号查找，或者根据姓名查找，也可以根据班级查找，把属于这个班级的学生全部显示；

（6）对学生信息根据成绩进行排序，指定一门课程，对学生进行排序。

2. 实现提示

（1）学生信息使用以下结构体：

```
struct student{
    int id;          //学号
    char name[32];   //姓名
    int age;         //年龄
    char sex;        //性别
    int class;       //班级
    float score[3]; //三门课成绩
};
```

（2）菜单操作实现使用以下代码：

```
#01: #include <stdio.h>
#02: #include <stdlib.h>
#03: int main(){
#04:     char ch;
#05:     int i;
#06:
#07:     do{
```

```
#08:        system("cls");
#09:        for(i=0;i<50;i++)
#10:           printf("*");
#11:        printf("\n");
#12:        printf("\t 1: 添加一个学生\n");
#13:        printf("\t 2: 删除一个学生\n");
#14:        printf("\t 3: 修改一个学生\n");
#15:        printf("\t 4: 查找一个学生\n");
#16:        printf("\t 5: 学生成绩排序\n");
#17:        printf("\t 6: 退出\n");
#18:        for(i=0;i<50;i++)
#19:           printf("*");
#20:        printf("\n");
#21:
#22:        do{
#23:           printf("\t 请选择输入选项[1-6]:");
#24:           ch=getchar();
#25:           fflush(stdin);
#26:           if ((ch>='1')&&(ch<='6'))
#27:              break ;
#28:        }while((ch<'1')||(ch>'6'));
#29:
#30:        switch (ch){
#31:           case '1': printf("Insert a student\n"); break;
#32:           case '2': printf("Delete a student\n"); break;
#33:           case '3': printf("Modify a student\n"); break;
#34:           case '4': printf("Find a student\n"); break;
#35:           case '5': printf("Sort score\n"); break;
#36:           case '6': exit(0);
#37:        }
#38:
#39:        system("pause");
#40:     }while(1);
#41: }
```

其中：

#02：该头文件实现#08 行和#39 行的功能。

#08：将输出窗口清屏，重新再输出窗口中显示内容。

#09～#20：打印菜单选项。

#22～#28：用户输入菜单选项，必须输入 1～6，否则重新选择。

#25：清除输入缓冲区，将第一次输入的回车清除掉，以免影响下次输入。

#30～#37：根据用户的选择，执行不同的功能，其中的 printf( ) 语句用函数代替。

#36：退出程序，等价于“return 0;”。

#39：程序暂停，输入任意字符后继续。

A.5.2 文件加解密系统

1. 要求

实现简单的文件加解密系统，保护文件内容，实现以下功能：

（1）密码保护，运行软件时必须输入正确密码才能进行操作；

（2）实现操作界面，通过一个简单的菜单选择相应的操作；

（3）加密文件，打开需要加密的文件，输入密钥，将加密的文件保存；

（4）解密文件，打开需要解密的文件，输入密钥，将解密的文件保存；

（5）修改密码，修改软件保护密码。

2. 实现提示

（1）密码保护可以使用变量保存初始保护密码，用户输入密码与初始保护密码进行比较，相同则校验通过，否则校验失败。修改了软件保护密码后，如果重新启动软件，则会重置为以前的密码。

（2）加密和解密使用同一个密钥，即采用对称加密算法。加密和解密需要相同的密钥，如果密钥不对，则无法成功解密。密钥为用户输入的字符串。

使用异或运算对文件进行加密解密，按照如下流程：①通过一次异或运算，生成密文，密文没有可读性，与原文内容不同；②密文经过一次异或运算，就会还原成原文，实现解密。

加密功能的部分代码如下：

```
#01: void encrypt(FILE *fp_src, FILE *fp_dest, char *key){
#02:     char buff[LEN];
#03:     int len,i;
#04:
#05:     while ((len=fread(buff,sizeof(char),strlen(key),
fp_src))>0){
#06:         for(i=0; i<len; i++)
#07:             buff[i] ^= key[i];
#08:         fwrite(buff,sizeof(char), len, fp_dest);
#09:     }
#10: }
```

其中：

#01：加密函数，fp_src 为要加密的文件指针，fp_dest 为加密后的文件指针，key 为密钥。

#02：buff 用于存放读取的文件内容，以及加密内容。

#05～#09：读取内容进行加密，然后保存到目的文件中。

#05：循环读取整个文件内容，每次读取到的长度为 len，保存在 buff 中。

#06, #07：对 buff 的内容进行加密（异或）运算。

#08：将 buff 的内容存入到目的文件 fp_dest 中。

附录 B　常用资料与 C 语言自测题

B.1　美国信息交换标准代码（ASCII）

表 B-1　美国信息交换标准代码（ASCII）

ASCII 值	控制码	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符
0	NUL	32	空格	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

表 B-2 ASCII 码中控制码含义

类 别	控 制 码	英 文 全 称	意 义
传送控制	ACK	Acknowledge	确认
	DLE	Date link escape	数据链路转义
	ENQ	Enquiry	询问
	EOT	End of transmission	传输结束
	ETB	End of transmission block	信息块传输结束
	ETX	End of text	文本结束
	NAK	Negative acknowledge	否认
	SOH	Start of heading	标题开始
	STX	Start of text	正文开始
	SYN	Synchronous idle	同步信号
格式控制	BS	Back space	退格
	CR	Carriage return	回车
	FF	Form feed	换页
	HT	Horizontal tab	横向制表
	LF	Line feed	换行
	VT	Vertical tabulation	垂直制表
	FS	File separator	文件分隔符
	GS	Group separator	组分隔符
	RS	Record separator	记录分隔符
	US	Unit separator	单元分隔符
其他	BEL	Bell	响铃
	CAN	Cancel	取消
	DC1	Device control 1	设备控制 1
	DC2	Device control 2	设备控制 2
	DC3	Device control 3	设备控制 3
	DC4	Device control 4	设备控制 4
	DEL	Delete	删除
	EM	End of medium	介质用毕
	ESC	Escape	退出、略过
	NUL	Null	空白、无效
	SI	Shift in	移入
	SO	Shift out	移出
	SP	Space	空格
	SUB	Substitute	代替，替换

B.2 运算符优先级

表 B-3 运算符优先级

	运 算 符	含 义	结 合 方 向
1	( )	括号、函数参数表	自左至右
	[ ]	数组元素下标运算符	
	→	指向结构体成员运算符	
	.	结构体成员运算符	
2	!	逻辑非运算符	自右至左
	～	按位求反运算符	
	++、—	自增 1、自减 1	
	+、−	求正、求负	
	*	指针运算符	
	&	取地址运算符	
	(类型名)	强制类型转换	
	sizeof	求所占字节数运算符	

续表

优 先 级	运 算 符	含　义	结 合 方 向
3	*、/	乘法、除法运算符	自左至右
	%	求余运算符	
4	+、−	加法、减法运算符	自左至右
5	<<、>>	左移、右移运算符	自左至右
6	<、<=	关系小于、小于或等于	自左至右
	>、>=	关系大于、大于或等于	
7	==、!=	关系等于、不等于	自左至右
8	&	按位与运算符	自左至右
9	^	按位异或运算符	自左至右
10	|	按位或运算符	自左至右
11	&&	逻辑与运算符	自左至右
12	||	逻辑或运算符	自左至右
13	? :	条件运算符	自右至左
14	=、+=、−=、*=、/=、%=、&=、^=、|=、<<=、>>=	赋值、运算并赋值	自右至左
15	,	逗号运算符	自左至右

B.3　常用库函数

（1）数值计算函数

数值计算函数在头文件 math.h 中。

表 B-4　数值计算函数

函 数 名	函 数 原 型	功　能
abs	int abs(int x);	求整数 x 的绝对值
fabs	double fabs(double x);	求 x 的绝对值
exp	double exp(double x);	计算 e^x 的值
pow	double pow(double x,double y);	计算 x^y 的值
sqrt	double sqrt(double x);	计算 x 的平方根，$x \geqslant 0$
fmod	double fmod(double x,double y);	求 x/y 的余数（双精度表示）
rand	int rand(void)	生成 0～32767 间的随机整数
floor	double floor(double x);	求不大于 x 的最大整数（双精度表示）
frexp	double frexp(double val,int*eptr);	把 val 分解尾数 x 和以 2 为底的指数 n，即 val=$x\times2^n$，n 存放在*eptr 中，返回尾数 x，$0.5 \leqslant x<1$
modf	double modf(double val,double *iptr);	把 val 分解成整数和小数部分，整数部分存放在*iptr 中，返回小数部分
log	double log(double x);	求 $\log_e x$，即 $\ln x$
log10	double log10(double x);	求 $\log_{10} x$，即 $\lg x$
sin	double sin(double x);	计算 sin (x)的值，x 为弧度
cos	double cos(double x);	计算 cos (x)的值，x 为弧度
tan	double tan(double x);	计算 tan (x)的值，x 为弧度
asin	double asin (double x);	计算 arcsin(x)的值，x 在−1～1 之间
acos	double acos (double x);	计算 arccos(x)的值，x 在−1～1 之间
atan	double atan (double x);	计算 arctan(x)的值

（2）字符函数

字符函数在头文件 ctype.h 中。除了 tolower 和 toupper 函数外，其他函数如果判断为真，则返回 1；否则返回 0。

表 B-5 字 符 函 数

函 数 名	函 数 原 型	功 能
isalpha	int isalpha(int ch);	判断 ch 是否为字母
isdigit	int isdigit(int ch);	判断 ch 是否为数字
isalnum	int isalnum(int ch);	判断 ch 是否为字母或数字
iscntrl	int iscntrl(int ch);	判断 ch 是否为控制字符
isspace	int isspace(int ch);	判断 ch 是否为空格、制表或换行字符
islower	int islower(int ch);	判断 ch 是否为小写字母
isupper	int isupper(int ch);	判断 ch 是否为大写字母
isgraph	int isgraph(int ch);	判断 ch 是否为可打印字符（ASCII 值 0x21～0x7e 之间）
isprint	int isprint(int ch);	判断 ch 是否为可打印字符（ASCII 值 0x20～0x7e 之间）
ispunct	int ispunct(int ch);	判断 ch 是否为标点字符（包括空格），即除字母、数字和空格以外的所有可打印字符
tolower	int tolower(int ch);	把 ch 中的字母转换成小写字母，返回小写字母
toupper	int toupper(int ch);	把 ch 中的字母转换成大写字母，返回大写字母

（3）字符串函数

字符串函数在头文件 string.h 中。

表 B-6 字符串函数

函 数 名	函 数 原 型	功 能
strlen	unsigned strlen(char *s);	求字符串 s 长度，不计最后的 '\0'
strcat	char *strcat(char *s1,char *s2);	把字符串 s2 连接到 s1 后面，返回 s1 地址
strcpy	char *strcpy(char *s1,char *s2);	把字符串 s2 复制到 s1 中，返回 s1 地址
strncpy	char *strncpy(char *dest, char *src, unsigned n);	复制字符串 s2 的前 *n* 个字符
strcmp	char *strcmp(char *s1,char *s2);	对 s1 和 s2 所指字符串进行比较。s1<s2，则返回–1，s1=s2，则返回 0，s1>s2，则返回 1
strcasecmp	int strcasecmp(char *s1, char *s2);	字符串进行比较时，自动忽略大小写的差异
strstr	char *strstr(char *s1,char *s2);	在字符串 s1 中，查找 s2 第一次出现的位置，返回找到的地址，找不到返回 NULL
strchr	char *strchr(char *s,int ch);	在字符串 s 中，查找字符 ch 第一次出现的位置，返回找到的地址，找不到返回 NULL

B.4 C 语言自测题

一、判断题（10 分，每题 1 分）。

1．表达式“7=3+4”正确。

2．“int c[][2]={{1,2},{3,4}};”是正确的二维数组定义语句。

3．yab 和 Yab 是不同的变量名。

4．字符数组定义为“char str[10]="abcde";”，该数组共有 5 个数组元素。

5．若变量 pointer 为指针变量，语句“pointer=2000;”是正确的赋值语句。

6．“int *p=&a, a;”是正确的变量定义语句。

7．用语句“scanf("%s",name);”对字符数组 name 进行输入操作，当输入“I_am　a　Student”时，数组的内容为“I”。

8．“while (1) { ;} ”表示无限循环。

9．在 C 程序中，当函数中所定义的局部变量与全局变量同名时，全局变量屏蔽局部变量。

10．指针变量本身没有指针。

二、选择题（40 分，每题 2 分）。

1．若定义数组：“int b[ ][3]={{1, 2, 3},{ 4, 5},{ 6, 7}}”，则 b[1][1]的值是（　　）。

A．5　　B．1　　C．7　　D．无确定值

2．变量名只能由字母、数字和下画线三种字符组成，且第一个字符（　　）。

A．必须为数字　　B．必须为字母

C．必须为字母或下画线　　D．可以是数字和下画线中的任一种字符

3．若有定义：“int a = 7; float x = 2.5, y = 5.7;”，则表达式“x + a % 3 + (int) (x + y)/4”的值是（　　）。

A．2.500000　　B．3.500000　　C．4.500000　　D．5.500000

4．已有定义：int a=7，则以下表达式中 b 的值不等于 2 的是（　　）。

A．b=a/3　　B．b=9–(a++)　　C．b=a%2　　D．b=a>3?2:1

5．若有说明：int a[]={1,2,3,4,5,6,7,8,9,10}, *p=a；则数组元素地址的正确表示（　　）。

A．&(a+1)　　B．&p　　C．&a[i]　　D．a++

6．设 x、y 是 int 型变量，且 x = 3、y = 4、z = 5，则下面表达式中使 x 值为 0 的是（　　）。

A．!((z < y)&&(x–=3))　　B．(x–y–1) || y

C．(z–y) && (!(x–=3))　　D．(y–x–1) &&(x–=3)

7．调用函数时，若实参是一个数组名，则向函数对应的形参传送的是（　　）。

A．数组的类型　　B．整个数组元素的值

C．数组第一个元素的值　　D．数组的首地址

8．逻辑运算符两侧运算对象的数据类型（　　）。

A．只能是 0 或非 0 正数　　B．可以是整型、字符型或实型数据

C．只可以是整型或字符型数据　　D．只能是 0 或 1

9．语句“for (x = 0, y = 0; (y = 123) || (x<4); x + +);”中，for 循环的执行次数是（　　）。

A．无限次　　B．4 次　　C．0 次　　D．3 次

10．下面程序段（　　）。

```
for ( t = 1; t < = 100; t++){
   scanf ("%d", &x);
   if ( x <0 )  continue;
   printf ("%3d", t );
}
```

A．当 x < 0 时，整个循环结束　　B．当 x ≥ 0 时，什么也不输出

C．printf( ) 函数永远也不执行　　D．最多允许输出 100 个正整数

11．以下程序中调用 scanf( ) 函数给变量 a 输入数值的方法是错误的，因为（　　）。

```
int main(){
    int  *p, *q, a, b;
```

```
    q=&a;
    p=q;
    printf("input a:");
    scanf("%d",*p);
…
}
```

A．*p 表示的是指针变量 p 的值

B．*p 表示的是指针变量 p 的地址

C．*p 只能用来说明 p 是一个指针变量

D．*p 表示的是变量 a 的值，而不是变量 a 的地址

12．下面程序的运行结果是（　　）。

```
a=1;  b=2;  c=2;
while (a<b<c) { c-- ;}
printf ("%d,%d,%d",a,b,c);
```

A．1, 2, 0　　B．2, 1, 0　　C．1, 2, 1　　D．2, 1, 1

13．对语句“int a[10] = {6, 7, 8, 9, 10};”的理解正确的是（　　）。

A．将 5 个初值依次赋给 a [0] 至 a[4]　　B．将 5 个初值依次赋给 a [1] 至 a [5]

C．将 5 个初值依次赋给 a [6] 至 a[10]　　D．因长度与初值个数不同，故语句错误

14．以下正确的函数声明语句形式是（　　）。

A．double　fun (int x , y)　　B．double　fun (int x; int y)

C．double　fun (int ,int) ;　　D．double　fun (int x; int y);

15．以下程序正确的运行结果是（　　）。

```
int main ( ){
    int  a=2,i;
    for (i=0;i<3;a++,i++)
    printf ("%4d",f(a));
}
f(int a){
    int  b=0;
    static  int  c=3;
    b++ ;
    c++ ;
    return (a+b+c);
}
```

A．7　7　7　　B．7　10　13　　C．7　8　9　　D．7　9　11

16．执行以下程序后，a、b 的值为（　　）。

```
int main(){
    int  a,b,k=4,m=6,*p1=&k,*p2=&m;
    a=p1!=&m;
    b=(-*p1)/(*p2)+m;
    printf("%d %d",a,b);
    printf("b=%d\n",b);
}
```

A．–1, 5　　B．0, 7　　C．1, 6　　D．4, 10

17．以下不正确的说法是（　　）。

A．在不同函数中可以使用相同名字的变量

B．形式参数是静态变量

C．在函数内的复合语句中定义的变量只在复合语句范围内有效

D．在函数内定义的变量只在本函数范围内有效

18．阅读下面程序，则输出结果是（　　）。

```
void fun(int a,int b){
     int c=20,d=25;
     a=c/3;
     b=d/5;
}
int main(){
     int a=3,b=5;
     fun(a, b);
     printf("%d,%d\n",a,b);
}
```

A．6, 5　　B．5, 6　　C．20, 25　　D．3, 5

19．在执行以下程序段时，（　　）。

```
x=-1;
do{
   x=x*x;
}while(x);
```

A．循环体将执行一次　　B．循环体将执行两次

C．循环体将执行无限次　　D．系统将提示有语法错误

20. 以下程序段在字符型变量 ch 的值为 'B' 时，（　　）。

```
switch (ch){
    case 'A':   printf("a");
    case 'B':   printf("b");
    case 'C':   printf("c");break;
    case 'D':   printf("d");
    default:printf("e");
}
```

A．输出：b　　B．输出：de

C．输出：abcde　　D．输出：bc

三、填空题（20 分，每空 2 分）。

1．以下函数的功能是：把两个指针变量所指向的变量的值进行交换。

```
void exchange(int *x , int *y){
     Int p, *t=&p;
     _________;
     *y=_________;
```

```
        *x=*t ;
}
```

2. 下面程序的输出结果是_____。

```
#include <stdio.h>
int f(int n){
     static int s=1;
     while(n)
        s*=n--;
     return s;
}
int main(){
     int i,j;
     i=f(4);
     j=f(2);
     printf("i=%d j=%d \n",i,j);
}
```

3. 以下程序的功能是计算数组之中的奇数之和，保存于变量 m，并输出结果。

```
int main(){
   int a[10]={3,4,5,6,1,2,3,4,5,7};
   int m=0, i;
   for(i=0; i<10; i++){
       if (________)
             m=________;
}
printf("m=%d", m);
}
```

4. 下面程序的输出结果是_____。

```
int main( ){
     Int a[10]={19,23,44,17,37,28,49,36}, * p;
     p=a;
     printf("%d\n", (p+=3)[2]);
}
```

5. 下面程序的输出结果是_____。

```
int main(){
     int i=0;
     while(i<7){
          if(i%3==0)
             printf("*");
          else
             printf("#");
          i++;
     }
}
```

6．下面程序的运行结果是_____。

```
void test(int *x, int *y){
     *x=9;
     *y=10;
}
int main(){
     int a=5, b=7;
     test(&a,&b);
     printf("%d,%d", a, b);
}
```

7．下列程序的输出结果是求 3×3 矩阵副对角线元素（a[0][2]、a[1][1]、a[2][0]）的和。

```
int main(){
      int a[3][3]={1,4,7,3,6,9,2,5,8},n,s=0;
      for(n=0;n<=2;n++)
          s=s+_________;
      printf("s=%d",s);
}
```

8．程序的输出为 CBA，填空完成输出语句。

```
int main( ){
      char *a="DCBA";
      printf("%s\n", _________);
}
```

四、改错题（10 分，每个错误 2 分）。

以下程序的要求是用函数求一个整型数组的平均值，有 5 个错误，找出并纠正错误，以“将#XX 行改为 YYY”的形式进行解答。

```
#1  float average(int n,array[]){
#2   int i;
#3       float aver;
#4       for(i=1;i<=n;i++)
#5           aver+=array[i];
#6       aver=aver/n;
#7       return ;
#8   }
#9
#10  int main( ){
#11      float aver;
#12      int i,score[];
#13      for(i=0;i<10;i++)
#14          scanf("%d",score[i]);
#15      average(score[10],i);
#16      printf("average=%6.2f\n",aver);
#17  }
```

五、编程题（20 分）。

1．实现一个函数 void outStr(char str[])，将数组 str 保存的一个字符串进行逆序输出。例如，字符串为“China”，则输出“anihC”。（8 分）

2．编写程序，建立一个一维整型数组，要求从键盘输入每个元素的值，然后将该数组中的整数从大到小进行排列。（12 分）

参考答案：

一、× √ √ × ×　× × √ × ×

二、ACDCC　CDBAD　DCACD　CBDCD

三、1．（1）*t=*y　（2）*x

2．（3）i=24, j=48

3．（4）a[i]%2!=0　（5）m+a[i]

4．（6）28

5．（7）*##*##*

6．（8）9,10

7．（9）a[n][2–n]

8．（10）a+1

四、1．将#1 行改为 float average(int n, int array[])

2．将#3 行改为 floart aver=0;

3．将#7 行改为 return aver;

4．将#14 行改为 scanf("%d",&score[i]);

5．将#15 行改为 aver=average(score,i);

五、略。

参 考 文 献

[1] Brian W Kernighan, Dennis M Ritchie. C 程序设计语言（第 2 版）. 徐宝文，李志，译. 北京：机械工业出版社，2004.

[2] Stephen Prata. C Primer Plus 中文版（第 5 版）. 云巅工作室，译. 北京：人民邮电出版社，2005.

[3] Ivor Horton. C 语言入门经典（第 5 版）. 杨浩，译. 北京：清华大学出版社，2013.

[4] K N King. C 语言程序设计：现代方法（第 2 版）. 吕秀锋，黄倩，译. 北京：人民邮电出版社，2010.

[5] 林锐. 高质量程序设计指南——C++及 C 语言（第 3 版）. 北京：电子工业出版社，2012.

[6] Kenneth A Reek. C 和指针. 徐波，译. 北京：人民邮电出版社，2008.

[7] Peter Van Der Linden. C 专家编程. 徐波，译. 北京：人民邮电出版社，2008.

[8] andrew koenig. C 陷阱与缺陷. 高巍，译. 北京：人民邮电出版社，2008.

[9] 维基百科. http://zh.wikipedia.org/.

[10] 互动百科. http://www.baike.com/.

反侵权盗版声明